Models and Simulations in the Semantic Web

Dissertation zur Erlangung des Grades

des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich–Technischen Fakultät
der Universität des Saarlandes

von Moritz Stüber

Saarbrücken, 2023

Tag des Kolloquiums

2023-12-21

Dekan

Prof. Dr. Ludger Santen

Mitglieder des Prüfungsausschusses

Prof. Dr.-Ing. Andreas Schütze	*Vorsitz*
Prof. Dr.-Ing. Georg Frey	*Gutachter*
Prof. Dr.-Ing. Mike Barth	*Gutachter*
Dr.-Ing. Sophie Nalbach	*akademische Mitarbeiterin*

This work © 2023 by Moritz Stüber is licensed under Attribution 4.0 International (CC BY 4.0).

To view a copy of this licence, visit `http://creativecommons.org/licenses/by/4.0/`.

In loving memory of

Thea Stüber
1923–2022

Abstract

Models of the dynamic behaviour of mechatronic systems and their simulation provide a helpful way of dealing with the systems' inherent complexity during both development and operations. However, the current state of technology has two significant shortcomings with respect to the use and reuse of models and simulations in a distributed setting: a lack of Findability, Accessibility, Interoperability, Reusability (FAIRness) and the heterogeneity of the models' interfaces. We explored the use of Semantic Web concepts and -technologies to provide modelling and simulation (M&S) entities and -capabilities as a service. Specifically, they were made available through a hypermedia application programming interface (API)—a service that fully implements the architectural style of the Web, Representational State Transfer (REST). This thesis shows that the approach makes sense; that it is new with respect to its specific focus; feasible; and directly useful. It is found that the approach increases the FAIRness of the exposed M&S entities and -capabilities and enables software agents to use them without being specifically programmed to do so on a syntactic level. Consequently, the problems caused by the interface heterogeneity of models are alleviated. In conclusion, the approach promotes the reuse of system models by multiple stakeholders for the development of systems as well as for the operations of distributed processes, and leads to interesting applications and further research.

Kurzfassung

Modelle des dynamischen Verhaltens mechatronischer Systeme sowie deren Simulation sind hilfreich, um die Komplexität der Systeme während Entwicklung und Betrieb beherrschen zu können. Allerdings werden mit dem aktuellen Stand der Technik zwei wesentliche Aspekte hinsichtlich Verwendung und Wiederverwendung der Modelle nicht ausreichend abgedeckt: die Auffindbarkeit, Zugänglichkeit, Interoperabilität und Wiederverwendbarkeit (FAIRness) sowie die Formenvielfalt der Modellschnittstellen. Wir haben die Anwendung von Konzepten und Technologien des Semantic Web für die Bereitstellung von Modellen und Simulationen als Service untersucht. Konkret wurden diese durch eine Hypermedia API bereitgestellt—einen Service, der den Architekturstil des Web, Representational State Transfer (REST), vollständig umsetzt. Die Dissertation zeigt, dass der Ansatz Sinn ergibt; sowie hinsichtlich seines Fokus neu; machbar; und direkt anwendbar ist. Es wurde demonstriert, dass der Ansatz die FAIRness der Modelle und Simulationen erhöht und Softwareagenten in die Lage versetzt, diese zu nutzen, ohne auf syntaktischer Ebene dafür programmiert worden zu sein. Letzteres ist eine Lösung für die aufgrund der Formenvielfalt der Modellschnittstellen entstehenden Probleme. Zusammenfassend fördert der gewählte Ansatz die Wiederverwendung von Systemmodellen durch verschiedene Interessierte für Entwicklung und Betrieb verteilter Systeme; und ist anschlussfähig durch wissenschaftliche Fragestellungen und Anwendungen.

Summary

Models of the dynamic behaviour of technical systems and their simulation are ubiquitous in engineering because they provide a helpful way of dealing with the complexity of today's mechatronic systems during both development and operations. Using and reusing models independently of the M&S specialists that originally created them is desirable because both development processes and the systems themselves are frequently distributed in nature, often across organizational boundaries. One example of this is ensuring that the requirements of a complex machine which is constructed using components provided by different manufacturers, such as the envisioned driving range of a battery electric vehicle (BEV), are met during development. Another example is creating forecasts for the generation and consumption of energy by technical systems, such as photovoltaic (PV) systems or energy storage systems, as an essential input to the control processes of smart grids. Additionally, models can be seen as both an asset because of the insight they can provide, and as an investment because their creation and validation takes expertise and time. This further stresses the importance of reusing them.

However, the current state of technology has two significant shortcomings with respect to the use and reuse of models and simulations in a distributed setting: a lack of FAIRness and the heterogeneity of the models' interfaces. The Findable, Accessible, Interoperable, Reusable (FAIR) principles outline a technology-independent path to improving the chances that both human *and* software agents learn about the existence and contents of digital assets in a way that enables them to decide whether they are relevant for their goals. Interface heterogeneity is inherent to models as each model exposes different parameters, inputs, and outputs. This results in manual programming effort for every model to be used—unless an alternative approach can be found.

Two additional aspects that ought to be improved are the coupling characteristics of interfaces and means to establish trust in models and simulation results. The coupling characteristics of the interface define in how far the independent evolution of individual systems is possible without breaking a system that is built as a combination of smaller systems. Because the adherence to exactly one, tightly specified interface cannot be enforced when the overall system is distributed across several organizations, it is desirable to support *loose coupling*. Not knowing the origins and scope of a model threatens its usability; therefore, measures that can inspire trust, such as providing metadata and provenance information, are desirable.

We explored the use of Semantic Web concepts and -technologies to provide M&S entities and -capabilities as a service. Specifically, models,

model instances, simulations and simulation results were made available through a hypermedia API, in other words a service that fully implements the constraints of the architectural style of the Web, REST. On the technical side, the work is based on the Functional Mock-up Interface (FMI) standard for model exchange and co-simulation. The technical work comprises devising and implementing ontologies and a parser to create representations of Functional Mock-up Units (FMUs) in the Resource Description Framework (RDF) data model; a cloud-native M&S service conforming to REST using recommended technologies for the Semantic Web; and use cases (UCs) to show feasibility and usefulness. Moreover, the Pragmatic Proof Algorithm (PPA) by Verborgh et al. was selected to demonstrate the use of the service by a generic software agent. It was implemented and extended to cope with requirements on requests that only become known at run-time. The implemented software is published under an open-source licence. Characteristics of the approach are analysed from an abstract systems engineering (SE) perspective, focusing on concepts and patterns instead of the proof-of-concept software developed as part of this work.

This thesis shows that the approach is reasonable in a number of ways; that it is new with regard to its specific focus and the technologies used; that it feasible; and that it can be immediately applied. This is done by motivating the concept in detail; describing its realization; demonstrating the functionality through UCs; reviewing the achieved characteristics; and putting them into context through a review of related literature.

It is found that the approach increases the FAIRness of the exposed M&S entities and -capabilities and enables software agents to use them without being specifically programmed to do so on a syntactic level. Consequently, the problems caused by the interface heterogeneity of models is alleviated. Furthermore, finding models through semantic queries is enabled. The desirable characteristics *loose coupling* and *horizontal scalability*, as well as providing provenance information, are supported as a result of the software's design.

In conclusion, the selected approach promotes the reuse of system models by multiple stakeholders for the development of systems as well as for the operations of distributed processes. It is feasible to realize the conceptual design choices with the selected technology stack. The UCs demonstrate that there are several cases in which the approach is useful.

It is expected that both useful direct applications and interesting research questions emerge as a result of this work. Possible direct applications include using the software as a building block for applications that require M&S capabilities in a service-oriented architecture (SOA); managing and developing collections of data sets and models centred around a topic, for example by research groups; and exploiting parallelization and load-dependent horizontal scaling when searching for optimal configurations. From an academic perspective, it would for example be interesting to explore ontology-driven modelling; the composability of models; and the potential of the approach for realizing useful digital twins.

Acknowledgements

Starting at the very beginning, this research project originated in me seeing a job offer on “Model and Simulation as a Service” as part of the project “Designetz” in the context of energy systems that feature large amounts of renewable and therefore weather-dependent energy sources. Curious about how modelling and simulation of technical systems, a major topic of my master’s in mechatronics at Vorarlberg University of Applied Sciences, might benefit from their provisioning *as a service*, I applied and got accepted. A time-consuming journey ensued, involving learning how to build cloud-native applications and how to represent knowledge in the Semantic Web; learning to balance project work, research, and teaching; as well as learning how to get your point across in meetings with people that have a wide range of professional backgrounds and agendas. While working on Designetz, ideas for a dissertation related to—but extending far beyond—the project deliverables formed.

Consequently, this work was partly supported by the SINTEG-project “Designetz” funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) under grant 03SIN224.

Prof. Dr.-Ing. Georg Frey ensured that I could indeed turn these ideas into *my* dissertation. I would like to express my gratitude for providing an environment in which following one’s curiosity and working independently, based on trust rather than formalities and pressure, was encouraged; and for reviewing intermediate results; publications; and this thesis.

Beyond Prof. Frey, the people at the Chair of Automation and Energy Systems (AES) at Saarland University also deserve a wholehearted “Thank you!”. I had the pleasure to collaborate with Lukas Exel, Florian Wagner, Christian Wolf, Christian Siegwart, Danny Jonas, Felix Felgner and Felix Scherhag. By discussing both major concepts and minor implementation details, I benefited from their knowledge and perspective. Josef Meiers knew everything about the inner workings of AES. Thank you, Rehan Khalid, Daud Minhas and Elham Abohamzeh for occasionally sharing your perspective on academia and everything else, brightening up everyday life in the office. Gisela Kempka and Robert Florange deserve a special thank you for administrative support; access to the workshop; and a genuine interest in the humans at AES and their occupations outside work.

A few people not related to systems engineering also helped with realizing this thesis and overcoming associated challenges. Klara proofread important texts and understands the highs and lows of pursuing a PhD. Svenja listened to my stories about working in academia and shared some of her own, while solving many bouldering problems of increasing difficulty. Luisa helped with developing strategies for balancing research and life on beautiful hikes in the mountains. Christian always finds the right words, even in difficult situations that emerge without warning. Thank you very much!

Last, but not least, I would like to thank my family for their unwavering support throughout the emotional roller coaster this project turned out to be; believing that I would eventually finish this thesis; and pushing me to continue working on it. Thank you, Nina, for seeing the good in me even when I could not, and being with me since we both decided that this is just too good to not try.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

AAA	anyone can say anything about any topic
ADS	architecture design space
API	application programming interface
BEV	battery electric vehicle
BPS	building performance simulation
CFD	computational fluid dynamics
CI/CD	continuous integration/continuous deployment
CLI	command-line interface
CNA	cloud-native application
CNS	cloud-native simulation
CoAP	Constrained Application Protocol
CPU	central processing unit
CSP	cloud service provider
CSV	comma-separated values
DAE	differential-algebraic equation
DEAP	Distributed Evolutionary Algorithms in Python
DeMO	Discrete-event Modeling Ontology
DevOps	software development and -operations
DIK	Data—Information—Knowledge
DIKW	Data—Information—Knowledge—Wisdom
DL	Description Logics
EOOL	equation-based, object-oriented modelling language
EYE	Euler Yet another proof Engine
FAIR	Findable, Accessible, Interoperable, Reusable
FAIRness	Findability, Accessibility, Interoperability, Reusability
FEM	finite element method
FLOSS	free/libre and open-source software
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FOL	first-order predicate logic
GA	genetic algorithm
HATEOAS	hypermedia as the engine of application state

HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure as Code
IDE	integrated development environment
iff	if and only if
IO	input-output
IOF	Industrial Ontologies Foundry
IP	intellectual property
IS	information science
JSON	JavaScript Object Notation
JSON-LD	JSON-based Serialization for Linked Data
KG	knowledge graph
LCIM	Levels of Conceptual Interoperability Model
LDP	Linked Data Platform
LOT	Linked Open Terms
LRU	least recently used
MBSE	Model-based Systems Engineering
MIMO	multiple-input/multiple-output
MSaaS	Modelling and Simulation as a Service
M&S	modelling and simulation
NATO	North Atlantic Treaty Organization
N3	Notation3
NWP	numerical weather prediction
OAS	OpenAPI Specification
OS	operating system
OSLC	Open Services for Lifecycle Collaboration
OWA	open-world assumption
OWL	Web Ontology Language
PDE	partial differential equation
PII	personally identifiable information
POA	plane of array
PPA	Pragmatic Proof Algorithm
PROV-O	The PROV Ontology
pURL	persistent URL
PV	photovoltaic
QPF	Quad Pattern Fragment
RDF	Resource Description Framework

RDFS	Resource Description Framework Schema
REST	Representational State Transfer
RIF	Rule Interchange Format
RMSE	root-mean-square error
RPC	remote procedure call
SaaS	Software as a Service
SE	systems engineering
SHACL	Shapes Constraint Language
SIMaaS	Simulation as a Service
SLR	systematic literature review
SMS	Systems, Models, Simulations
SNAF	Scoped Negation as Failure
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
SOSA	Sensors, Observations, Samples, Actuators
SPARQL	SPARQL Protocol and RDF Query Language
SPDX	Software Package Data Exchange
SSP	System Structure and Parameterization
TTL	time to live
TPF	Triple Pattern Fragment
UC	use case
UI	user interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VCS	version control system
WAMS	Web Architecture for Modelling and Simulation
WBS	Web Based Simulation
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition
YAML	YAML Ain’t Markup Language

1. Introduction

Can modelling and simulation benefit from Semantic Web concepts?

Both research fields, modelling and simulation (M&S) and the Semantic Web, concern themselves with formalisms for representing and sharing knowledge as well as algorithms for deriving new information from what is already known. This research argues that combining concepts and tools from both fields by providing models, simulation results and the ability to perform simulations in the Semantic Web, is logical and promising because the underlying ideas and approaches to solving problems are similar. Additionally, it is argued that this combination is also novel with regard to the specific focus and concepts chosen; feasible; and directly useful.

1.1 Context and Basic Idea

In the context of this work, the term *modelling* refers to the collection of relevant knowledge about certain aspects of a system in the form of physical laws expressed using mathematical equations. A *model* is the unambiguous representation of this knowledge in a *formal language*, that can thus be acted upon by *algorithms* executed on a computer. The numerical approximation of the set of model equations by means of an algorithm given start values and boundary conditions is called *simulation*. For such dynamic system models, the result of the simulation is a trajectory of values over time.

In this section, basic terms and ideas are briefly defined informally, without literature references, to provide an overview and to enable the formulation of high-level research questions. Detailed definitions and references follow in chapter 2.

Despite the features of equation-based, object-oriented modelling languages (EOOLs) that facilitate the reuse of model code for building hierarchical models within the confines of the modelling language ecosystem, and despite the existence of widely supported model exchange formats such as Functional Mock-up Interface (FMI), the *Findability, Accessibility, Interoperability, Reusability (FAIRness)* of models in general is poor and hinders the reuse of the knowledge that the models encode.

The *Web* is a vast collection of interlinked documents or web pages made accessible on the *internet*, a communication network of computers around the world. The Web contains massive amounts of information which can be accessed by human users because they are capable of intuitively navigating web pages by following links to achieve their goals. Moreover, human users typically understand the consequences of actions on the Web without executing them—for example, that clicking a button labelled “buy now” in a web shop most likely results in the exchange of the user’s money for things or services offered through the web shop.

The degree to which the information present on the Web can be used by *software agents* to solve tasks without human intervention is limited because they lack the aforementioned abilities. Today, *Web application programming interfaces (APIs)* are the means by which functionality is exposed to software on the Web and the web pages humans see are often merely views on the functionality provided through the underlying Web APIs. However, using a Web API can be compared to memorizing the exact Uniform Resource Locators (URLs) and mouse movements required to achieve a certain task on a website and then executing them blindfolded: a change in a URL or the layout or structure of a page breaks the application. Moreover, it is impossible for today's software agents to answer certain types of questions such as "which model was simulated to show that a design meets this requirement and which parameter set was used?"; to discover related content that is not explicitly listed; and to reason about the provided information and infer new information based on what is already known. *Therefore, it is argued that the full potential of the Web cannot be realized unless software agents are enabled to do so.*

The *Semantic Web* describes an extension of the Web which provides information about the semantics of data and the semantics of API interactions in a machine-readable form. Essentially, this requires a shared language for representing the meaning and interrelations of things, namely the Web Ontology Language (OWL), as well as a distributed data model that uses this language to specify the semantics of data and its relation to other data, namely the Resource Description Framework (RDF).

Performing work using software running on a digital computer requires implementing algorithms using a programming language and representing the data the algorithms operate on in a machine-processable format. The term *software architecture* denotes the internal structure of such programs. Typically, software is written in a layered fashion by extensively reusing established software libraries and even entire programs to build an application. Thus, defining the software architecture involves choosing a suitable *technology stack* for the task at hand.

Comparable to the categorization of buildings based on their architectural style, *architectural styles* in software design have been identified based on commonalities of specific software architectures. Certain architectural styles have been found to adequately account for the requirements of a class of systems with similar characteristics. Two architectural styles relevant to distributed software systems are *REST* and *SOA*: Representational State Transfer (REST) has been identified as the architectural style underlying robust and scalable distributed hypertext systems such as the Web; and service-oriented architecture (SOA; an architectural *style* despite its name) has been found suitable to realize immensely complex and high-performing applications on the Web, such as Netflix or Amazon. To emphasize that SOAs work best when services are responsible for a single task only and the overall functionality is achieved by combining services, the term *microservices* is often used. Typically, microservices are realized as REST-based HTTP-APIs, in other words APIs that are accessed via Hypertext Transfer Protocol (HTTP) and that are designed based upon—but do not fully implement—the constraints defined by REST.

For systems of systems that are distributed across organizational boundaries and for which there is thus no central authority that can enforce exactly one way of creating interoperable interfaces, it is desirable to strive for *loose coupling* between systems. Loose coupling facilitates the independent evolution of subsystems and thereby the robustness of the overall system as well as its ability for growth.

If and only if (iff) all constraints that define the architectural style REST are realized in a specific software, then the software supports its use in loosely coupled systems. Web APIs that fully implement REST are called *hypermedia APIs* because the term “RESTful” is frequently misused.

In addition to facilitating the creation of loosely coupled systems, hypermedia APIs also facilitate their use by software agents that are not specifically programmed for a certain API, which are called *generic* or *intelligent* software agents. One task that these agents need to solve is which requests need to be sent in which sequence in order to reach a goal, given a set of hypermedia APIs. For this, the *Pragmatic Proof Algorithm (PPA)* by Verborgh et al. can be used.

The discipline and activity of identifying requirements, defining a software architecture capable of satisfying these requirements and implementing software to realize a certain functionality is denoted *software engineering*. A systematic approach to software engineering and thoughtful design of a software’s architecture are immensely helpful when attempting to create large, complicated software systems such as a suite of microservices providing M&S-capabilities in the Semantic Web. Consequently, software engineering represents an important part of this work.

The design and realization of applications that run “in the cloud” is a question of software engineering. *Cloud computing* describes the desired characteristics for providing software *as a Service* over the internet, as well as the practice of doing so. *Cloud-native applications (CNAs)* are applications that are specifically designed to run in the cloud and consequently realize the defining characteristics.

To summarize, the research presented in this thesis is an interdisciplinary endeavour at the intersection of three fields of research: through *software engineering*, capabilities of *modelling and simulation* as a branch of systems engineering (SE) shall be made available in the *Semantic Web*, which is a branch of computer science.

1.2 Research Questions and Thesis Outline

The core idea of this thesis is that a hypermedia API as an exemplary interface to RDF data; the PPA as an example of a generic software agent; and the realization of the hypermedia API as a CNA could be suitable design choices to address the lack of FAIRness and machine-actionability of M&S resources. Specifically, complete and valid system models in the form of Functional Mock-up Units (FMUs); their simulation; and the corresponding simulation results are considered. This idea raises four questions:

Q1. Can the FAIRness of M&S entities and -capabilities improve by providing them through a hypermedia API that exposes RDF representations of its resources?
Q2. Does this hypermedia API enable the use of M&S capabilities by an implementation of the PPA as an example of a generic software agent?
Q3. Does this hypermedia API support its use in loosely coupled systems?
Q4. Which other benefits or drawbacks arise from the chosen approach?

In order to answer these questions, a concise explanation of the fundamental concepts is provided in chapter 2. Based on definitions for data, information, knowledge, and FAIRness, triple-based and equation-based formal representations of data and knowledge are introduced. Then, quality attributes for (distributed) software systems are summarized and the main concepts of the Web and the Semantic Web are introduced. Last, aspects of software engineering that are relevant to this thesis are briefly summarized.

In chapter 3, the academic literature on the combination of concepts from M&S with the Web and the Semantic Web is reviewed; resulting in the formulation of a research gap.

Based on issues with the reusability of M&S resources observed with current tooling and the formulation of desirable characteristics, the hypotheses that detail the research questions are developed in chapter 4. The combination of concepts chosen for demonstrating that the approach is feasible and useful is explained in a technology-independent manner. A conceptual overview is given to summarize the intended interplay of concepts and technologies.

The design and implementation of all software components, such as the hypermedia API that exposes M&S resources and -capabilities in the Semantic Web, is presented in chapter 5. This includes the design of two ontologies and a parser necessary to represent and reason about the chosen model format (FMI) and systems, models and simulations on a more abstract level using the RDF data model. Moreover, the Quad Pattern Fragment (QPF) interface; RESTdesc; and the PPA as necessary building blocks to validate the hypotheses of this work are explained.

Next, applications that demonstrate that the chosen approach can be useful for finding models, simulating models in the cloud and integrating models and simulations in a SE knowledge graph (KG) are summarized in chapter 6.

In chapter 7, the research questions and hypotheses are evaluated, and possible directions for further research are outlined. Chapter 8 provides a summary and concludes the thesis.

The appendices A and B contain additional details, such as a glossary (table A.1), and a list of publications and software created in the context of this thesis (appendix B.1.)

Many interesting aspects and questions are out of scope for this thesis. Specifically, this includes...

- *other definitions of the terms modelling and simulation*—only the definitions outlined above, which reflect the use of the terms in the context of working with the EOOL Modelica, apply;
- *computational fluid dynamics (CFD) and finite element method (FEM) models*—from a mathematical point of view, only systems that can be represented using differential-algebraic equations (DAEs) (not partial differential equations (PDEs)) are in scope;
- *co-simulation* and the use of other modelling languages and exchange formats—only the use of Modelica and FMI for modelling and simulating the dynamic behaviour of technical systems using a single solver per simulation are regarded;
- *digital twins*—an investigation of the aspects specific to models designed to accurately represent and interact with real systems over their entire life cycle is out scope; and
- *user interfaces* for human users including concepts and tools for the visualization and analysis of simulation results.

1.3 Contributions

From the author's point of view, the following contributions to the scientific discourse on providing and using M&S capabilities in distributed systems of systems were made:

▼ [94, section 6.2]

- it was shown that the chosen approach can increase the FAIRness and improve the machine-actionability of M&S capabilities in a way that supports loose coupling, and reasons for why these goals are desirable were given;
- a proof-of-concept implementation of a hypermedia API that exposes M&S functionality in the Semantic Web was published as open-source software, and a detailed description of design concept and implementation was provided;
- the FMI-ontology and the `fmi2rdf`-parser used to build representations of FMUs in the RDF data model were made available openly;
- an extension of the PPA to support user input which has requirements that only become known at run-time was suggested; and
- an open-source implementation of the PPA as well as an example demonstrating its robustness against changes in the service interface were published.

▲

1.4 About This Document

This document contains text and figures that have already been published as part of scientific papers written for communicating aspects of the work presented.

This practice is known as *text recycling*, which is defined as

> "[...] the reuse of textual material (prose, visuals, or equations) in a new document where (1) the material in the new document is identical to that of the source (or substantively equivalent in both form and content), (2) the material is not presented in the new document as a quotation (via quotation marks or block indentation), and (3) at least one author of the new document is also an author of the prior document" [38]

Reasons for recycling text include

- describing recurring aspects of research, such as the description of methods used, *consistently*;
- avoiding superficial changes that are intended to disguise an existing text as a new one, such as rearranging sentences and using synonyms;
- improving the quality of the document by reusing text that has already been proofread by reviewers and co-authors.

However, text recycling can be problematic iff is it is not disclosed to readers (falsely implying novelty); breaks contracts with publishers; or creates an unfair advantage in a competitive setting.

To mitigate these problems, best practices for text recycling are suggested:

- Hall, Moskovitz and Pemberton [38] stress that text should be recycled *ethically and appropriately* (only if accurate and effective in new context; to ensure consistency); *legally* (compliant with copyright agreements); and *transparently* (proactively and openly communicated).
- Meinel [61] highlights that the lack of attribution is what causes most problems and makes specific suggestions on *how* to alert readers to recycled text in different circumstances.

It is attempted to realize these suggestions in this document. Specifically, ...

- this section communicates that parts of this thesis can also be found in other scientific publications of which I am the first and corresponding author;
- in appendix B.1, there is a list of these publications including comments pointing out their relation to each other and to this thesis;
- throughout the document, longer sections of text (possibly including tables or figures) that are recycled from other documents are pointed out through margin notes (as shown to the left).

▼ [..., section x.y]

▲

It is possible that minor edits to the recycled passages were made to ensure good readability.

2. Fundamental Concepts

“Model-based Systems Engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases” [98]. Alternatively, Zeigler, Mittal and Traore [126] more briefly state that “Model-based Systems Engineering employs model-based practices to engineer IT-enabled systems”.

But what are systems, what are models, and what is their relation to data, information and knowledge? Cellier and Kofman [21, section 1.2] define a *model* as the result of “extraction of knowledge from the physical plant to be simulated, organizing that knowledge appropriately, and representing it in some unambiguous fashion”. *Systems* are seen as a part of the world that is currently of interest, defined by separating it from its environment through the definition of inputs and outputs [20, section 1.1]. From a different angle, systems can also be seen as “a potential source of data”; leading to the definition of *experiments* as “the process of extracting data from a system by exerting it through its inputs”. In turn, *simulation* then “concerns itself with performing experiments on the model to make predictions about how the real system would behave if these very same experiments were performed on it” [21, section 1.2].

In this chapter, definitions for *data, information and knowledge* are provided and the so-called *FAIR principles* are summarized. Then, two ways to represent knowledge in a way that allows computing new information from what is already known, namely *equation-based* and *triple-based*, are introduced.

Moreover, *quality attributes for distributed software systems* such as *dependability and security, trust* and *loose coupling* are discussed followed by outlining the core concepts and technologies of the *Web* and the *Semantic Web*. Last, *cloud computing, SOA, and DevOps* are briefly introduced as they represent aspects of software engineering relevant to this thesis.

2.1 Data—Information—Knowledge and FAIRness

Three concepts fundamental to the scientific field information science (IS) as well as to this thesis are *data*, *information* and *knowledge*. However, there are many definitions of these concepts that vary in their approaches, content and applicability in computer science as well as their inclusion or omission of related concepts such as ‘signals’ and ‘wisdom’. Typically, data, information, knowledge and wisdom are seen as hierarchical concepts and are thus denoted *Data—Information—Knowledge—Wisdom (DIKW) pyramid* or *wisdom hierarchy*. Higher levels in the hierarchy are associated with increasing structure, meaning and context, but decreasing actionability by machines. Levels are often defined with respect to lower levels but the transformation processes from one level to another are not always clear.

Rowley [82] analyses definitions provided in textbooks relevant to information systems and knowledge management. She retraces the definitions by pointing out commonalities and differences and discusses key aspects in terms of the relationships and transformation processes as well as the variables that change between concepts [82, section 6]. Her conclusions are that even though there is consensus about the hierarchical nature of the concepts, the distinction between them needs further clarification, among other dimensions with respect to whether or not a concept can be represented in systems or solely exists in the minds of humans. Furthermore, it is noted that wisdom is discussed in significantly less detail than data, information, and knowledge despite its position at the top of the DIKW pyramid.

Zins [127] states and analyses the definitions of data, information and knowledge provided by 45 expert researchers in the field of IS. In his own definitions [127, p. 486] and his identification of different models for defining Data—Information—Knowledge (DIK) [127, p. 488], he offers an answer to whether or not information and knowledge can only exist in the human mind by distinguishing between the subjective domain and the universal or collective domain. *Subjective knowledge* refers to the knowledge of and inseparable from an individual and can contain both private and public elements. In contrast, *universal* knowledge denotes knowledge that exists independent of individuals. Zins provides the following definition of DIK in the universal domain: "in the objective domain **data** are sets of signs that represent empirical stimuli or perceptions, **information** is a set of signs, which represent empirical knowledge, and **knowledge** is a set of signs that represent the meaning (or the content) of thoughts that the individual justifiably believes that they are true", prepended by the statement that "data, information, and knowledge are human artefacts [...] represented by empirical signs" [127, p. 487]. This definition emphasizes that while the distinction between DIK is a construct of humans, sets of observable signs can be used to represent DIK in the universal domain. Also, it sees information as a type of knowledge that is *empirical*, in other words "derived from or guided by direct experience or by experiment, rather than abstract principles or theory" [23].

In alignment with Zins' definition and its underlying view that DIK exist both in the subjective and the universal domain, I use the terms Data—Information—Knowledge as follows within the context of this work:

Anything related to *wisdom* is out of scope.

Data denotes representations of observations or properties of things *without* a context that gives them meaning. Things can be the source of data if they are observed, and they can be both real-world entities such as a specific photovoltaic (PV) system or virtual entities such as a model that represents it. Examples for data include a cell in a table, or an arbitrary key/value-pair in a JavaScript Object Notation (JSON) object. The sets of signs used to encode data are unambiguous so that they can be acted upon algorithmically.

Metadata

is data that "describes, annotates, or gives information about other data, including but not limited to tags in a programming code, information about a digital file's characteristics, or a library catalogue showing the location and call number of books" [24].

Information

is an intentional selection of data within a context and therefore meaningful to an agent. It can be seen as the answer to a question expressed as a subset of the available data and thus as empirical knowledge. For example, information could be a row, column or single cell of a table including the context of what kind of data is stored in the table and what the respective column headers and/or row indices are; it could the value of a specific key in a JSON document; or a set of triples about an entity.

Knowledge

is seen as a representation of something that an agent justifiably believes to be true within a context. In contrast to information, representations of knowledge can include non-empirical data such as general interrelations between concepts or specific relationships between individuals. Examples for such knowledge include an ontology that defines terms to describe a domain of interest as well as their relations; a representation of a general concept inclusion; or a system of equations describing the dynamic behaviour of a technical component. Knowledge allows inferring new data from what is already known, for example by means of reasoning or simulation.

The purpose of this clarification on the three terms is to enable a concise description of the scope of the technological building blocks described in following sections. It can also serve as one criterion for discussing the possible value of the solutions developed using these building blocks.

Another criterion for evaluating the properties of the devised solutions are the so-called *"FAIR Guiding Principles for scientific data management and stewardship"*. In the initial publication of these principles [125], the authors motivate and propose 15 specific requirements for maximizing the value of data sets through facilitating reuse, especially in a scientific context.

FAIR stands for *Findable, Accessible, Interoperable, Reusable*. The FAIR principles are suggested as a high-level answer to what constitutes good data management that is independent of a specific domain or implementation. The scope of the principles explicitly includes research output in general, for example information about steps leading to the creation of data, such as procedures, (software) tools and algorithms [125, p. 1].

The target audience for FAIR data comprises both human users and software agents. Wilkinson et al. motivate their emphasis on machine-actionability by an increasing need for software support for finding, retrieving and analysing data relevant to a certain purpose due to the ever-increasing volume, complexity and variety of formats of data [125, p. 3].

Machine-actionability is seen as the ability of a software agent to make correct and meaningful choices when faced with data that it has never encountered before [125, pp. 3]. In order to achieve this, software agents need to be enabled to identify the structure and intent of the data in question; to decide whether it is useful for the task at hand and only subject to constraints that can be met; and finally to learn about their options for acting on the data. Information on these subtasks provided in a well-known format is what enables their solution; consequently, machine-actionability is not either realized completely or not realized at all, but realized gradually subject to the amount of information useful to software agents that is available. This is emphasized by the fact that this information could be provided for none, either, or both of contextual metadata and the data itself.

In table 2.1, the individual FAIR guiding principles are listed along with their nature (technical (t), organizational (o), (meta)data modelling (m)) and an association of each principle to one or more of the following dimensions:

D1: (Meta)data Modelling
These principles are mostly about what kind of data should be provided and what high-level features need to be supported by the formalisms used to do so. Note that making a data set FAIR means that metadata and links to other data need to be curated and provided *in addition to* the data itself.

D2: Alignment with Semantic Web Technologies
Principles marked with D2 align well with the Semantic Web because the requirements they suggest are realized by technologies that are part of the Semantic Web software stack (compare section 2.3.3). For example, A1, A1.1 and A1.2 demand that resources should be available using a universally implementable protocol that is openly standardized, free and supports authentication. Hypertext Transfer Protocol Secure (HTTPS) as one of the pillars of the Web and the Semantic Web fulfils these demands.

D3: Responsibility for Ownership
D3 is intended to point out that there are principles which require special care and a sense of responsibility on the side of the provider in order to achieve the intended consequences. As examples, consider A2 and the use of *persistent* identifiers (F1), which mandate a long-term commitment to the continued FAIRness of a data set.

Wilkinson et al. hope that by providing data in a FAIR manner, not only users but also the creators and other stakeholders such as institutions or funding agencies benefit: in addition to its vastly improved findability, FAIR data can be cited and thus provides a measure for its contribution to the scientific discourse and recognition for the effort put into creation and publication. For a given data set, better FAIRness would likely lead to a higher impact and, seen from an economical perspective, better return on investment.

The GO FAIR initiative was formed to promote the creation of FAIR data sets by communicating their potential; training stakeholders; and building standards, best practices and tooling to support researchers. In January 2020, broad support for the goals of the GO FAIR initiative was expressed by representatives of major research networks by signing the “Sorbonne declaration on research data rights” [91].

Id	Principle	Nature	D1	D2	D3
F1	(Meta)data are assigned a globally unique and persistent identifier.	t, o	◆	◆	◆
F2	Data are described with rich metadata (defined by R1 below).	m	◆		◆
F3	Metadata clearly and explicitly include the identifier of the data they describe.	m	◆		
F4	(Meta)data are registered or indexed in a searchable resource.	o			◆
A1	(Meta)data are retrievable by their identifier using a standardised communications protocol.	t		◆	
A1.1	The protocol is open, free, and universally implementable.	t		◆	
A1.2	The protocol allows for an authentication and authorisation procedure, where necessary.	t		◆	
A2	Metadata are accessible, even when the data are no longer available.	o, t			◆
I1	(Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.	t	◆	◆	
I2	(Meta)data use vocabularies that follow FAIR principles.	o, m	◆	◆	
I3	(Meta)data include qualified references to other (meta)data.	m	◆		
R1	(Meta)data are richly described with a plurality of accurate and relevant attributes.	m	◆		
R1.1	(Meta)data are released with a clear and accessible data usage licence.	m	◆		◆
R1.2	(Meta)data are associated with detailed provenance.	m	◆		◆
R1.3	(Meta)data meet domain-relevant community standards.	m	◆		◆

Table 2.1: List of the 15 FAIR Guiding Principles. A detailed explanation of the principles including examples and hints regarding implementation can be found on the website `https://www.go-fair.org/fair-principles`. The content of the columns ‘Id’ and ‘Principle’ are copied verbatim from this website; the other columns are added by me.

2.2 Formal Representations of Data and Knowledge

▼ [94, section 1]

Today’s industrial systems are complex mechatronic systems, integrating mechanical, electrical and computational elements. In this context, formal models are used that describe the dynamic behaviour of systems by means of equations. From a mathematical point of view, a system of differential-algebraic equations (DAEs) is created that implements the laws of physics, supported by empirical data such as look-up tables if a physics-based modelling approach is infeasible. The approximation of this system of DAEs by means of numerical integration algorithms is called simulation. For models of the dynamic system behaviour, the result of a simulation is a trajectory of values over time.

In the context of the Semantic Web, formal models are ontologies. Ontologies encode concepts, roles, and their interrelations; computational reasoning is the process by which satisfiability, classification, axiom entailment, instance retrieval et cetera are computed.

Both types of models—ontologies and systems of DAEs—are a useful tool for the design, analysis and understanding of complex systems. Importantly, they are also *abstractions* of the domain of interest, meaning that a distinction between relevant and irrelevant aspects with respect to the intended purpose of the model was necessarily made by the humans who created the model. Moreover, the models have to be encoded in a formal language such as the Web Ontology Language (OWL) for ontologies or Modelica for DAEs in order to enable algorithms to operate on them.

As a consequence of the choice for a specific modelling language, a limit in scope and expressivity is imposed on the modelling process. This means that the types of problems that can be solved using the chosen language, including its ecosystem such as model libraries, integrated development environments (IDEs) and expert communities, are limited.

The limit that a modelling language imposes is not a problem if the modelling task at hand fits the capabilities of the language well. However, it is an interesting question whether two modelling approaches with different purposes and capabilities can be meaningfully combined in order to support the investigation of other types of problems, drawing on the respective strengths of the individual approaches and alleviating their disadvantages.

▲

This section explains the fundamentals of equation-based and triple-based representation of knowledge as used in systems of DAEs and ontologies, respectively.

2.2.1 Equation-based Representation of Knowledge

▼ [94, section 1.1]

In engineering, models and simulations are ubiquitous and used in many steps of a product's lifecycle. Reasons to use modelling and simulation (M&S) [20, p. 10 f.] [84, p. 4] include that the feasibility of a design with respect to requirements, safe operating conditions, and the sizing of components can be evaluated. "What if?"-questions can be analysed faster and safer than if they were executed on real systems. Moreover, models allow access to internal states that could not be measured easily in reality, and they also allow the computational search for optimal configurations. During development, models facilitate the parallel development of different parts of a system by different people, such as different physical components; the physical component and its control strategy; or training operators before the real system becomes available. During operations, M&S can be used for fault detection, as a virtual sensor, or as an essential building block for realizing the ideas for optimizing a system's behaviour summarized under the term "digital twin".

For coping with the multitude of questions to be answered by M&S, different approaches exist: finite element method (FEM) and computational fluid dynamics (CFD) methods are used to analyse phenomena that vary both over time *and* location, such as the distribution of mechanical stress inside a component or the flow of air around an object. Event-based approaches are used for queueing situations or crowd simulations; agent-based approaches can for example model economic questions and other interactions between distinct entities with different agendas.

In this work, we focus on dynamic models for the time-varying behaviour of quantities in technical, multi-domain systems that can be represented as a system of DAEs. This focus is consistent with clusters of competence in industry and academia and caters to a large, relevant class of problems. Other uses of the terms modelling and simulation are equally valid in their respective contexts, but out of scope for this work.

However, even within this scope, despite the similarity of the underlying mathematical problems to solve and as a consequence of both different requirements for different applications and historical reasons, many formalisms and corresponding ecosystems exist today. They range from the use of general-purpose programming languages (such as Python); via modelling languages explicitly designed to support multi-domain models as well as to support language features that facilitate the development of robust, well-structured models; to highly specialized languages and tools for specific applications. ▲

In order to assist users with managing the complexity inherent to modelling cyber-physical systems, modelling languages such as *Modelica* [66] provide mechanisms for structuring larger models as a hierarchical collection of submodels that can be developed independently. Specifically, the mechanism that enables the use of independently developed submodels or components, is the so-called *acausal* formulation of the model, meaning that the question "which variable gets calculated in which equation, and in which order do the resulting assignments need to be evaluated?" is answered by an algorithm instead of the human creating the model. Moreover, state-of-the art modelling languages provide mechanisms for reusing parts of the model code and facilitating the development of *model libraries*, for example object-orientation.

▼ [94, section 2.1]

The first step in creating a model is finding an abstraction of a system that is suitable for the purpose of the model. In-depth domain knowledge is required in order to understand and describe the *relevant* properties of the system at hand. Second, the model needs to be implemented in a formal modelling language, which requires expertise in the modelling formalisms, languages and algorithms used because attention to the numerical properties of the model and the use of structuring mechanisms that ensure the reusability of the model must be paid. M&S environments such as `Dymola`[1] provide support regarding the implementation of the models by checking for syntactic mistakes, providing access to model libraries, offering the possibility to graphically create models by assembling component models, translating the models to executable form, triggering their execution and allowing users to analyse the results. However, it remains the user's responsibility to make sense of the calculated trajectories and to ensure that they are valid. In practice, obtaining accurate parameter values and measurement data for verification and validation of developed models represents a major challenge. ▲

▼ [94, section 2.1]

Unfortunately, models encoded in different languages are generally not interoperable. *Interoperability* denotes the degree to which systems can work together; the different "levels that need to be aligned in order to make systems meaningfully interoperate with each other" can be expressed using the Levels of Conceptual Interoperability Model (LCIM) [106, p. 6]. For the combination of models, *conceptual interoperability* (the highest level of interoperability according to the LCIM) is required. Through the term conceptual interoperability, it is expressed that the abstractions made in the creation of the models must align in order to get meaningful output from the combined models. In other words, a "state ensuring the consistent representation of truth in all participating systems" is necessary, which is the definition of *composability* suggested by Tolk [106, p. 7]. ▲

▼ [94, section 1.1]

The lack of interoperability is problematic for several reasons. From a practical perspective, the lack of interoperability hinders and slows down

[1] `https://www.3ds.com/products-services/catia/products/dymola/`

the development of complex systems that use components by other manufacturers, such as cars. From an economic perspective, the lack of reusability resulting from the limited interoperability is also problematic because models can represent valuable assets: the creation and validation of models requires resources, time and expertise which can be a significant investment. The value of this investment is maximized if the model is reused often, also outside the context for which it was originally created.

To solve the problem of syntactic interoperability, the Functional Mock-up Interface (FMI) standard [65] was developed. FMI is a tool-independent, open standard for model exchange that defines the interface, capabilities and format of so-called Functional Mock-up Units (FMUs), which is the name for models that are compliant with the FMI standard. There are two variants: FMUs for model exchange only contain the model equations and require an external solver for simulation. In contrast, FMUs for co-simulation contain a solver and can thus be used both as a standalone executable form of a model as well as in conjunction with other FMUs for co-simulation. In this work, co-simulation is out of scope; only the simulation of a single model/FMU with a single solver is considered.

▲

From a technical point of view, FMUs are archives containing a descriptive Extensible Markup Language (XML)-file, platform-specific binaries, C-code and optional additional files stored as a .fmu-file. FMI is widely adopted and supported by more than 150 tools to varying extent[1].

2.2.2 Triple-based Representation of Knowledge

One way to convey the meaning of a construct such as a series of characters, for example "s, e, n, s, o, r", is to simply acknowledge that this construct is a *symbol that refers to a thing in the world*, in this case a sensor (referential semantics) [1, p. 31]. In the context of the (Semantic) Web (section 2.3.2, section 2.3.3), things in the world are seen as *resources*; consequently, a way to make statements about resources is required.

The RDF Data Model and -Serializations

A more detailed introduction to RDF is provided in [87].

The Resource Description Framework (RDF) is a data model that allows making statements about resources. These statements always take the form of simple sentences consisting of a subject, an object, and a predicate that connects the subject to the object.

Predicate: "one of the two main parts of a sentence, containing the verb and any of its objects, modifiers, or other completions, and generally expressing an action, state, or condition" [25]

These so-called subject–predicate–object *triples* result in the data structure of a *directed graph* with labelled edges. As an example, consider the graph visualized in figure 2.1. It contains nodes (visualized as circles) connected by edges that have a direction (arrows); both nodes and arrows can be named. Moreover, a dashed box around a node denotes that the node is an instance of a class; and there are some literals (values), such as the string "°C". In tabular form, the same graph would look like this:

[1] `https://fmi-standard.org/tools/`

Subject	Predicate	Object
`10704`	is a	weather station
`10704`	hosts	`t_2m`
`t_2m`	is a	sensor
`t_2m`	observes	air temperature at 2 m above ground
...	...	...

As shown, the graph could imply the following: weather station `10704`, attributed to Germany's National Meteorological Service (DWD), hosts a sensor `t_2m` that observes the air temperature at 2 m above ground. The latest measurement taken by this sensor reveals that the temperature was 10.3 °C; the measurement process is also associated with DWD.

Each triple asserts something that is believed to be *true*. However, strings like "10704" which label nodes and edges carry no inherent meaning; if at all, they only make sense locally/to the person who created the graph. This means that data from different sources could not be merged automatically; but, since the Web is distributed, the data model needs to account for this.

By using Uniform Resource Identifiers (URIs) for subjects, predicates and objects (objects can be literal values, too), they become globally unique. For example, consider changing the sentences "`10704` is a weather station" and "`10704` hosts `t_2m`" as follows:

```
<https://example.com/weather-stations/10704>
<https://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<https://dbpedia.org/resource/Weather_station> .

<https://example.com/weather-stations/10704>
<http://www.w3.org/ns/sosa/hosts>
<https://example.com/sensors/t_2m> .
```

Now, anyone can say anything about any topic (AAA) [1, p. 35] and when the same URIs are used, they refer to the same thing. Furthermore, different data sets can be merged by simply concatenating them.

When the URIs are also Uniform Resource Locators (URLs) (compare section 2.3.2), their meaning/definition can be looked up. For example, `https://www.w3.org/1999/02/22-rdf-syntax-ns#type` indicates that the subject is an instance of a class indicated by the object of the triple, and the documentation for `http://www.w3.org/ns/sosa/hosts` tells us that this predicate expresses the "relation between a Platform and a Sensor, Actuator, Sampler, or Platform, hosted or mounted on it".[1]

To export, transfer and import graph data structures used in software, serialization formats are needed. The simplest serialization for RDF is called N-Triples [6] (shown above).

To reduce verbosity and size and to facilitate reading by humans, alternative syntaxes have been defined. One used in later chapters is called Turtle [7]. Most prominently, Turtle allows abbreviating URIs using the

[1] `https://www.w3.org/TR/vocab-ssn/#SOSAhosts`

`@prefix` and `@base` keywords; also, objects of triples that share the same subject and predicate can be listed separated by commas and predicates and objects connected to the same subject can be listed separated by semicolons (listing 2.1). The letter `a` further abbreviates `rdf:type`. In lines 21 to 24, a *blank node* is used as the object. Blank nodes are nodes that do not have a globally unique identity.

An RDF document can contain zero or more *named graphs* in addition to exactly one default graph [57, section 4]. For serializing the resulting subject–predicate–object–graph *quads*, alternative syntaxes such as N-Quads [17] (similar to N-Triples) or TriG [13] (similar to Turtle) are required. In TriG, named graphs are enclosed in curly braces.

Using graphs to store assertions about entities that belong to a domain of interest results in *knowledge graphs (KGs)*. KGs are defined as "a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities" by Hogan et al. [45, p. 2].

Querying graphs to retrieve answers to semantic queries, in other words finding entities based on their relations to other entities, is enabled through the SPARQL Protocol and RDF Query Language (SPARQL) [41].

Ontologies based on RDFS and OWL

In the running example, RDF has so far been used to make statements about specific entities (instances of classes). However, it is also possible to express relationships between classes, or properties of predicates; for example: `sosa:Observation rdfs:subClassOf prov:Activity`. *Class expressions*, *predicate expressions* and *individual assertions* (an entity is an instance of a class; two instances are linked through a predicate; two entities are equal/not equal) form the basis of *ontologies*.

The terms *concept* and *role* are sometimes used instead of *class* and *predicate*.

▼ [94, section 2.1]

Ontologies are consistent specifications of concepts relevant to a domain of interest and their interrelations in a formal language. In addition to this conceptual representation of knowledge, in other words being a "model of" some domain with the intent to facilitate its *description*, ontologies are also a "model for" systems to be built and are thus of *normative* nature too [44].

With respect to *what* is modelled by an ontology, we follow the conceptualization of Hofmann, Palii and Mihelcic [44, p. 136] and distinguish between *methodological* and *referential* ontologies. Methodological ontologies describe ("model of") methods or formalisms such as FMI, which are usually consistent and free of conflicting definitions of concepts. This facilitates their modelling as an ontology and as a result, the ontology has a high potential for adoption in implementing systems (normative aspect, "model for") [44, pp. 136, 138 f.]. In contrast, referential ontologies attempt to model what is and what is not important to describe a part of the real world, which generally represents a more diverse, inconsistent and ambiguous domain than a human-made concept such as a modelling formalism. Consequently, referential ontologies are less likely to be reused outside their original context [44, pp. 136, 143].

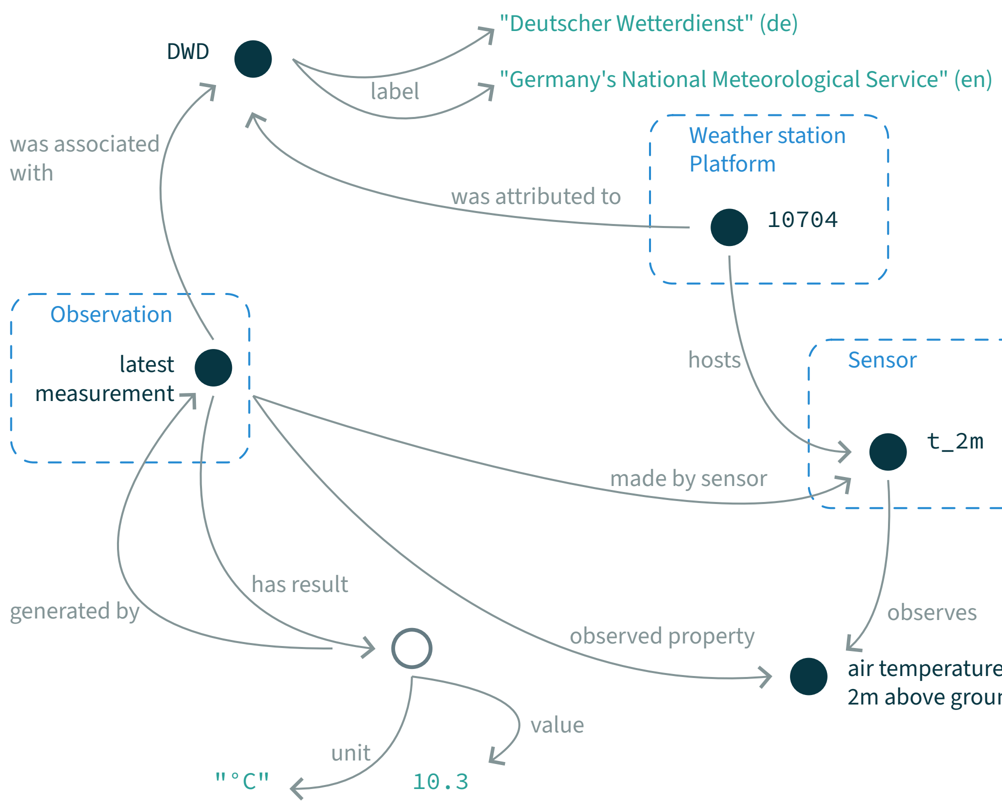

Figure 2.1: Statements about a resource, in this case a weather station, visualized as graph

```
@prefix rdf:  <https://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbr:  <https://dbpedia.org/resource/> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix unit: <http://qudt.org/vocab/unit/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@base <http://example.com> .

</weather-stations/10704> a dbr:Weather_station, sosa:Platform ;
    sosa:hosts </sensors/t_2m> ;
    prov:wasAttributedTo </agents/dwd> .

</sensors/t_2m> a sosa:Sensor ;
    sosa:observes </weather/t_2m> .

</measurements/latest> a sosa:Observation ;
    prov:wasAssociatedWith </agents/dwd> ;
    sosa:madeBySensor </sensors/t_2m> ;
    sosa:observedProperty </weather/t_2m> ;
    sosa:hasResult [
        qudt:numericValue 10.3; qudt:unit unit:degC ;
        prov:wasGeneratedBy </measurements/latest>
    ] .

</agents/dwd> rdfs:label "Deutscher Wetterdients"@de ,
    "Germany's National Meteorological Service"@en .
```

Listing 2.1: Statements about a resource, serialized using Turtle

The value of ontologies in the domain of M&S is expected to manifest itself by facilitating knowledge exchange and reuse; by helping with the resolution of compatibility questions; through their support for reasoning; and their role in querying data sets with respect to their semantics [44, p. 138 f.]. Specific mechanisms by which these are facilitated include the precise definition of terms; the resolution of ambiguity of terms through namespacing; serving as a consistent and shared (mental) model used by researchers in a topic area; and the ontologies' foundation in formal logic [110, p. 68 f.].

For the Semantic Web, Resource Description Framework Schema (RDFS) [14] and the Web Ontology Language (OWL) [86] provide the modelling constructs (class and role definitions using URIs) to implement ontologies that capture knowledge about entities and their interrelations in RDF.

Reasoning under the Open-world Assumption

The class and predicate assertions as well as equality or inequality statements that are made *explicitly* when creating a representation of an entity in RDF are in general *incomplete*. This is because it cannot be known a priori which statements will be needed and which ontologies are expected to encode them, in other words which path through a graph will be taken when answering queries; and because it simply does not make sense to manually add assertions and statements if and only if (iff) they can be derived via reasoning. For example, one might be interested in which agent the platform that the sensor `t_2m` is hosted by, is attributed to. In SPARQL, this query could be formulated as follows:

```
SELECT ?agent WHERE {
  <t_2m> sosa:isHostedBy ?platform .
  ?platform prov:wasAttributedTo ?agent .
  ?agent rdf:type prov:Agent .
}
```

However, neither the triples in the first line nor in the third line of the query are contained in listing 2.1. Taking into account additional class- and predicate expressions, they can however be deduced.

```
prov:wasAttributedTo rdfs:range prov:Agent .
sosa:hosts owl:inverseOf sosa:isHostedBy .
```

The first triple entails that `</agents/dwd> rdf:type prov:Agent` and from the second line follows that

```
</sensors/t_2m> sosa:isHostedBy </weather-stations/10704> .
```

This is called *reasoning*. With reasoning, the query can be resolved.

At its core, reasoning allows clients to infer facts from other facts. Common reasoning tasks [83, section 7] include deciding whether...

- a KG is free of contradictions/consistent (satisfiability);

– a statement is true given a KG (axiom entailment);
– a class can have instances (concept satisfiability);
– retrieving instances of a class from the KG (instance retrieval); and
– calculating the hierarchy of classes (classification).

OWL is based on Description Logics (DL), which is the name for *decidable* fragments of first-order predicate logic (FOL) [83, section 1.2]. Multiple DLs exist, which differ in the supported modelling constructs and, consequently, in their expressivity and scalability/computational complexity [83, pp. 79, 96]. Likewise, OWL *profiles* were created with the intention to provide subsets of OWL that are particularly suited for specific tasks (such as satisfiability and class hierarchies; query answering; or scalable reasoning); that are, in contrast to OWL 2 Full, decidable; and that keep the computational complexity within reasonable bounds at the expense of limiting the allowed modelling constructs [see 69, section 5].

A comprehensive introduction to Description Logics is provided in [83].

One important characteristic of DL and, consequently, OWL and its subsets, is the so-called open-world assumption (OWA). Under the OWA, facts which cannot be deduced from a KG are seen as *unknown* [83, section 6.10]. However, sometimes it is useful to assume that something not explicitly stated to be true is *false*. Therefore, mechanisms to do so were proposed—including Scoped Negation as Failure (SNAF), for example as part of Notation3 (N3) [81, section 3.10.4], and RDF Surfaces [43].

2.3 Successful Distributed Software Systems

It was estimated that in 2022, 5.3×10^9 people or 66 % of the world's population used the *internet* [59]; likely, they also used the *Web*.

Although frequently used interchangeably, there is an important distinction to be made between the internet and the Web[1]. The internet is "the global system of interconnected computer networks that uses the Internet protocol suite [...] to communicate between networks and devices" [124], whereas the Web is "an *information space* in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers" [46, emphasis added]. This means that the entire Web is on the internet; but also that there are applications on the internet that are not part of the Web (such as e-mail).

Considering its size, its effect on people's lives, and the fact that the Web already exists for more than 30 years, it can undoubtedly be seen as an immensely successful distributed software system.

In explaining the success of such systems, identifying their *architectural style* is helpful. An architectural style is "a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style" [30, section 1.5]. The architectural style of the Web is called Representational State Transfer (REST).

Humans cannot process data at the same speed and volume as machines. However, the Web is primarily intended for humans using a browser

[1] `https://www.w3.org/help/#webinternet`

to access its contents. The *Semantic Web* is an extension of the Web, adhering to the same architectural style, that enables software agents to support humans in achieving their goals.

In this section, work on general quality attributes for software systems; the Web and REST; the Semantic Web; and intelligent software agents is briefly summarized with the intent to facilitate understanding of the following chapters.

2.3.1 Quality Attributes for Software Systems

Intuitively, software users expect that a program returns correct results to their input within a reasonable amount of time. Furthermore, it is expected that a user interface remains responsive to input and that there are no unintended side effects. However, even seemingly simple programs are in fact quite complex: a programming language depends at least on the corresponding compiler or interpreter, but typically, many additional software libraries are used when developing a program using a programming language. These libraries in turn may have dependencies on their own and programs get released in versions, of which several might be used in practice even though only the latest version should be used. The existence and use of such a hierarchical collection of libraries enables quick development of high-level functionality, but it makes it effectively impossible for developers to fully understand every detail of what they create. In theory, it might be possible (but laborious) to read all the source code iff only open-source libraries are used; but as soon as closed-source components are used, it becomes actually impossible to understand a program completely.

Consequently, one can only hope that the components used to create a program are of high quality. But what does 'high quality' mean for software, and what are criteria for evaluating software quality?

From a user's perspective, *functionality, user interface* and *perceived performance* suggest themselves, but they are only sufficient iff there are only very minor consequences should they not be met. For example, this might be the case for a free, open-source game that does not require or collect any information about its users and does not require or use a connection to the internet. As soon as there are more severe consequences in case of failures, either because the user invests into a software in the form of continued commitment to its use or the payment of license fees, or because the software processes data that is not intended to be shared with the whole world, such as personally identifiable information (PII), a software should also provide its functionality in a *dependable and secure* manner. Moreover, *interoperability* with other systems, in other words the "ability to exchange information and to use the data exchanged in the receiving system" [107, p. 686] is often desired and seen as a facet of software quality.

From a provider's perspective, changes in either the program's environment and dependencies or the program itself are unavoidable if a program is not provided for a single use only, but for recurring use over a period of time. Therefore, it is desirable to systematically avoid or at least reduce problems caused by these environmental changes, for example by aiming for *loose coupling* with third-party dependencies. Problems can also occur if a program is intended to be used by several remote users and there is no adequate strategy for realizing *scalability* (the ability to increase performance, possibly beyond the resources of a single computer) and *elasticity* (the ability to scale up and down according to demand).

The main aspects of dependability and security as the core indicator for software quality; the concept of trust as accepted dependence; as well as loose coupling as a concept central to the architectural style of the (Semantic) Web are summarized in more detail below.

Trust

Reusing content provided by others raises the question of trust: do we believe that an interaction with something that we place our trust in will work out despite the uncertainty inherent to the interaction? If so, to what extent and why?

Sources for the uncertainty of the outcome of an interaction include *remote* access to resources; capabilities *provided by others*; and the existence of *several possible alternatives*. First, for resources only accessible remotely, there is a risk that malicious agents or network failures alter or inhibit the transfer of content over the network or that the behaviour of remote services simply doesn't match the expectations. Moreover, resources may only be available to a limited audience, either for functional reasons or because some clearance is needed to access them, which requires identification. By some means, trust in claims of identity and access rights must be established. Second, using content provided by others requires trust simply because it is out of one's own control: claims could be wrong or incomplete and their origin and possible underlying assumptions are likely intransparent to users. Third, any search resulting in more than one result requires choosing between options. These options might only be provided by different entities, but they might also present differing or even contradicting data. Evaluating the options with regard to their trustworthiness offers one strategy for choosing between them.

In their 2007 survey on trust in computer science [3], Artz and Gil identify and summarize four directions of research: *Policy-based* approaches apply rules to the existence, nature and/or absence of credentials in order to establish trust in the context of a specific interaction. In contrast, *reputation-based* approaches infer trustworthiness from a history of prior interactions and/or experience by others. Strategies for deciding on *trust in information resources* include explicit provenance information, content analysis and analysis of metadata such as hyperlinks and site design. Work on *general models of trust* is intended to foster understanding of the concept "trust" itself and its applicability in computational contexts. At a very abstract level, a distinction between basic trust over all contexts, general trust between individuals independent of the context and situational trust between individuals in a specific context as different types of trust is made. Artz and Gil [3, section 7] conclude that an evaluation of trust comprises the questions of which entity's trustworthiness is evaluated (target); how trust is represented (representation); how trust is evaluated (method), managed (management) and computed (computation); and what this evaluation is used for (purpose).

One way to evaluate the trustworthiness of an entity could be to evaluate its *dependability and security*.

Dependability and Security

Avizienis et al. [4] identify, define and explain attributes of dependable and secure computing; provide a categorization of threats to dependability and security; discuss general means to mitigate these threats; and highlight meaningful relations between the concepts. They provide two definitions of dependability [4, section 2.3]: first, it is seen as "the ability to deliver service that can justifiably trusted" with trust, in turn, being defined as "accepted dependence". Second, dependability of a system can also be seen as "the ability to avoid service failures that are more frequent and more severe than is acceptable".

Dependability comprises five primary attributes: availability, reliability, safety, integrity and maintainability [4, section 2.3]. Likewise, security is also a composite concept comprising the primary attributes availability, integrity and confidentiality. Because dependability and security share the aspects availability and integrity, they are seen as related concepts that should be considered jointly when developing and operating software.

The term 'service' will be introduced in detail in section 2.4. For now, a service can be seen as a synonym of 'software'.

When a desired functionality is provided by a service as intended, it operates correctly as observable through its external state, in other words at its service interface. A *service failure* occurs iff the external state exposes incorrect behaviour. Service failures are the result of *errors* propagated to the service interface that are caused by *faults* [4, section 3.5]. In other words, faults manifest themselves as errors which may lead to service failures iff they affect the external state of a program.

Avizienis et al. [4, section 3.2] identify 16 elementary fault classes and 31 likely combinations thereof that can, on a high-level, be grouped in development faults, physical faults and interaction faults. Examples of these fault classes include *external human-made malicious deliberate faults during operation* such as intrusion attempts (interaction); *internal human-made non-malicious, non-deliberate development faults due to incompetence* such as allowing direct, unguarded use of text provided by users because the developer is unaware this represents a serious risk (development); or *internal natural non-malicious non-deliberate faults during operation* such as the failure of a hard disk (physical).

Severe development faults could lead to complete or partial development failures, meaning that a developed software is put to use only with limited functionality, too late, or not at all due to the developers incapability to meet budget and/or schedule constraints or because even though a specification is implemented correctly, the specification did not capture the desired functionality.

In case the software does get used, failures can be categorized with respect to their domain (for example wrong content or wrong timing); detectability (signaled/unsignaled); consistency (reproducible, non-reproducible); and the severity of their consequences (from minor to catastrophic).

There are four basic strategies for minimizing the existence and consequences of faults on the dependability and security of services: first, the use of development methodologies, associated tools and general engineering strategies such as modularization is aimed at *fault prevention*. Second, mechanisms to detect and, if possible, recover from errors so that they do not lead to failures are *fault tolerance* techniques. Examples would be the automatic rollback to an earlier version in case severe failures occur after deploying an update; or automatic switching to a battery as power supply in case of a grid outage. *Fault removal* [4, section 5.3] as the third strategy comprises identification and removal of faults both during development as well as during use. During development, *verification* denotes the processes used to make sure that an implementation matches its specification ; whereas *validation* denotes the evaluation in how far the implemented software realizes its intended functionality, which may or may not be captured by the specification. Fault removal during use represents two out of four forms of maintenance, namely corrective maintenance in case the fault to be removed was active and detected, and preventive maintenance in case the fault was dormant but still recognized. The other forms of maintenance are adaptive maintenance (reaction to changes in the software's environment) and augmentative maintenance (adding of new features) [4, figure 3]. The fourth basic strategy against threats to dependability and security is *fault forecasting*. Fault forecasting can comprise both qualitative and quantitative evaluations of a specific service in question, as well as general knowledge about typical threats or expert opinions.

Verification is typically done using *tests* when developing SaaS, but more formal approaches such as model checking could be used in general.

In practice, the mental framework provided by the article by Avizienis et al. can be used to perform a *dependability and risk analysis*. This analysis combines fault forecasting and fault removal techniques with the intent to confidently arrive at the conclusion that a dependable service can be delivered [4, section 5.5]. Typically, applicable risks to the primary attributes of dependability and security are identified and quantified in a way that accounts for both their entry probability and the severity of the consequences in the event that a fault leads to a failure: the higher the probability and the higher the consequences, the higher the value associated with a specific risk. Usually, this analysis results in a set of *dependability and security provisions* to be followed during development and operations because a risk is seen as acceptably low iff a countermeasure is taken and as too high otherwise. Here, best practices such as those outlined by "IT-Grundschutz"[1], should be followed because they cover the most important aspects and can therefore even be useful in case a full dependency and security analysis cannot be performed due to insufficient resources and/or capabilities.

This identification could for example be based on the OWASP Top 10 in combination with experience made in the target environment for a software.

In an ideal world, software would be free of faults. However, the second definition and the notion of trust as accepted dependence as well as the explanations in section 5.5 of [4] emphasize that implementing a fully dependable and secure software is impossible. The result of analysing dependability and security and specifying measures aimed at achieving it therefore can only lead to *fault acceptance*, which is ideally based on a quantified estimation and therefore justifiable.

[1] https://www.bsi.bund.de/EN/Topics/ITGrundschutz/itgrundschutz_node.html

For the software presented in this thesis, a detailed upfront consideration of dependability and security are out of scope because the main purpose of the software is to demonstrate feasibility (proof of concept). However, this doesn't mean that dependability and security are disregarded: straightforward measures such as input validation are usually implemented and adding more elaborate countermeasures such as authentication are supported by design, but not implemented. Details on the strategies adopted with the intent of aiding in the creation of dependable and secure software can be found in section 5.3.3.

Loose Coupling

There are two approaches to designing a system of systems with regard to how interoperability shall be achieved. The first approach is to enforce exactly one world view across all interfaces of individual systems. By limiting the choices system developers can make to those allowed by a central entity, efficient and safe connections between a limited number of systems within a closed context [76, pp. 913, 919] can be realized. The interfaces follow a logically consistent pattern and any given interface is either compliant or not compliant with the specification. However, defining the scope and specifics of such a centralized design is likely non-trivial and, more importantly, it makes adaptive maintenance harder because of the high number of interdependencies: if a breaking interface change becomes necessary, then all systems need to update their implementation; ideally, all at the same time. This problem gets worse if changes in the environment are incompatible with the assumption on which the enforced world view is based.

The second approach is to acknowledge that in general, developers are faced with "an open world of uncertainty and conflicting concepts" [76, p. 913] and to therefore minimize the assumptions made about others. Inevitably, this leads to systems of less constrained and more diverse systems. However, because only minimal interdependencies between systems exist, changes in individual systems should only have limited effects on other systems, therefore facilitating a more independent evolution that is cheaper and contributes to better scalability.

The term *loose coupling* is used to denote the positive aspects of the second approach. Comparable to dependability and security, it is defined in terms of different facets. Pautasso and Wilde [76] provide definitions for each facet and characterize what constitutes loose and tight coupling for each of the facets. They also point out that loose coupling is not inherently better than tight coupling even though it is frequently used as if it were in arguments about system design—instead, each approach represents a decision to be made consciously when designing the software architecture for solving specific problems [76, p. 919].

Intuitively, the Web's open and diverse nature as a global system of systems with millions of stakeholders could be seen as an argument for aiming for loose coupling when designing for the Web. Alternatively, the relatively loose coupling promoted by the architectural style of the web [76, p. 919] can be seen as the reason for its success; suggesting that all software developed to become part of the (Semantic) Web should also aim for loose coupling. Either way, the facets put forth by Pautasso and Wilde [76] in their analysis of loose coupling are relevant for this work because they can serve both as a guideline during development as well as as criteria for evaluating relevant concepts and the implemented software.

Figure 2.2 graphically summarizes the facets of coupling and the possible extent of each facet. The facets are applicable to architectural styles in general; to specific system architectures; and to the interaction between two specific systems. A distinction is made between loose coupling (closest to the center), design-specific coupling and tight coupling (farthest from the center). The notion of 'design-specific coupling' allows meaningful analyses of architectural styles or architectures which do not unambiguously define the realization of a facet [76, p. 918]. For example, the architectural style REST does not specify the granularity of exposed resources, meaning that whether or not a specific RESTful implementation is tightly or loosely coupled with respect to granularity cannot be said in general.

From figure 2.2 and the explanations by Pautasso and Wilde [76] follow requirements for the design and implementation of loosely coupled systems. First, it must be possible for loosely coupled systems to discover each other by referral instead of through a centralized registry. With regard to the information model, the facet *(data) model* implies that messages must not be serializations of a shared application-specific data model, but instead documents that do not make any assumption about the receiver except that it can read the serialization format. Furthermore, it is suggested that less service interactions required to achieve a goal, facilitated by more extensive message contents, lead to more loose coupling (*granularity*). The facets *interaction*, *state* and *conversation* further specify that loosely coupled systems must support the asynchronous exchange of stateless messages that allow discovering the correct sequence of interactions in a conversation *at run-time*. Stateless messages are messages for which the receiver does not require knowledge about the history of prior interactions in order to unambiguously interpret it. When coupling software systems (services), each service has an internal representation of data and internal identifiers for accessing specific entities and functionalities. In loosely coupled systems, *global* names such as URIs are used for *identification* at the service interface; and the *binding* from an internal representation of a functionality to its specific implementation is done as late as possible. From the principle of late binding follows that loosely coupled systems do not make use of static *code generation* at design-time or of libraries that provide a programming language-specific interface to an application programming interface (API) instead of calling the API directly (*interface orientation*). With regard to *evolution* of their service interface, services that aim to be part of loosely coupled systems take special care not to introduce breaking changes. Last, loosely coupled systems are not required to run on the same *platform*, which could be ensured by using the openly standardized technology of the Web in many cases, but might be more difficult in cases where the individual systems are either incapable of using this stack or unwilling to.

To conclude, if the realization of an application as a composition of several systems can be seen as a source of faults threatening its maintainability

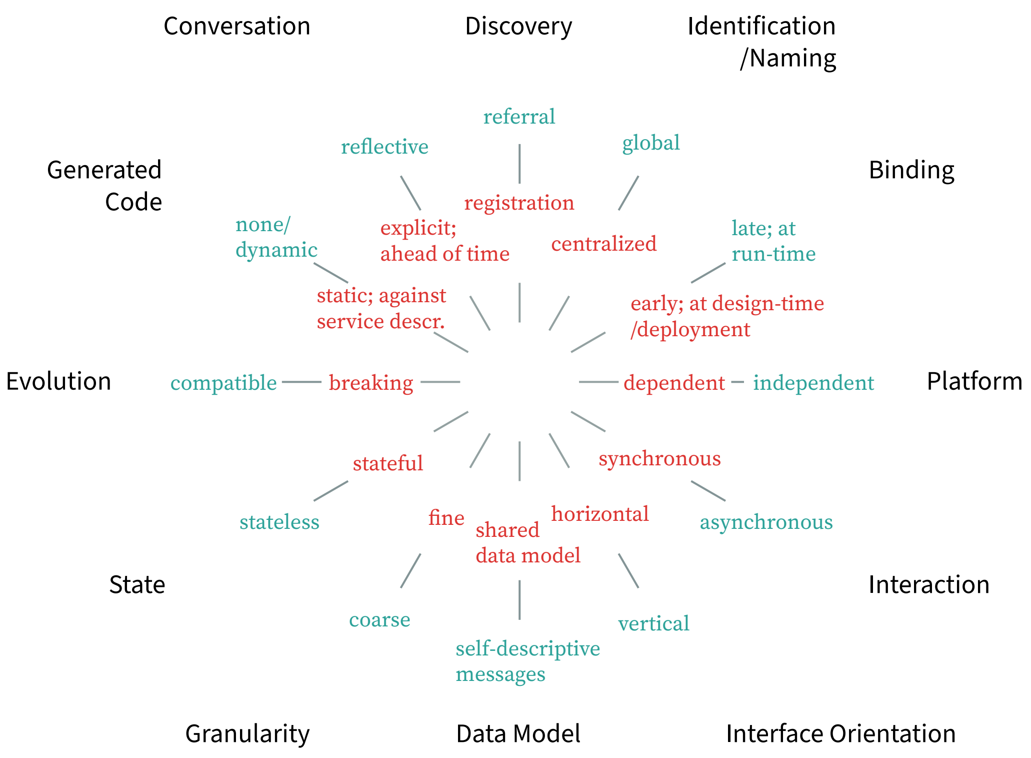

Figure 2.2: Facets of Coupling in Systems of Systems

and availability (especially if parts of the resulting system of systems are operated by others), an architecture realizing loose coupling can then be seen as design choice aimed at minimizing the occurrence and implications of these faults. One such architecture is that of the Web—its architectural style REST can be implemented such that loose coupling is fully supported [76, p. 919].

2.3.2 The Web

An excellent introduction to the Web, adopting a much broader point of view, is freely available through the slides of the *Web Fundamentals* course by Ruben Verborgh.

From a technical point of view, the Web is defined through three complementary technologies that constitute its software architecture: Uniform Resource Identifiers (URIs) and Uniform Resource Locators (URLs); Hypertext Transfer Protocol (HTTP); and hypertext.

- A URI is "a compact sequence of characters that identifies an abstract or physical resource" [9]. URL "refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network 'location')" [9, section 1.1.3].
- HTTP is "a stateless application-level protocol for distributed, collaborative, hypertext information systems" [32], used for retrieving representations of resources identified and located through URLs.
- Hypertext is understood as "the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions. Hypermedia is just an expansion on what text means to

include temporal anchors within a media stream; most researchers have dropped the distinction." [31, comment 3].

Fielding, one of the authors of both the URI and HTTP specifications, analysed the architectural style underlying the Web that, in combination with its consequently open and collaborative nature, made its success possible. He denoted it *Representational State Transfer (REST)* [30].

Representational State Transfer

From a practical perspective, REST can be roughly summarized as follows (see [118] for a detailed explanation): a service exposes a set of conceptual resources. These resources are identified and located by URLs. As a reaction to the application of HTTP verbs (`GET`, `POST`, ...), a *representation* of the resource (typically hypertext/hypermedia) is returned. Clients navigate between resources by selecting controls provided in the resource representations.

▼ [94, section 1.2] Technically, REST can be realized using other protocols such as Constrained Application Protocol (CoAP). ▲

REST applies to client-server systems, based on the idea that the resulting separation of concerns facilitates scalability and longevity. The constraints characterizing REST concern the identification of resources, the manipulation of resources through representations, the exchange of self-descriptive messages and the use of hypermedia as the engine of application state (HATEOAS); they are intended to result in a *uniform interface* [118, section 3.3].

▼ [92, section 3.2] ▲

▼ [92, section 3.2] ▲

- Services adhering to REST give access to a set of *resources* that are uniquely identified by URIs. Resources are unique conceptual entities of the service interface; the semantics of a resource always remains the same. Conceptual entities of the application domain are mapped to resources when defining the service functionality. Depending on the meaning of a resource, it is possible that this mapping changes over time. For example, a resource that represents the most recent version of a virtual representation of a specific system may point to different model instances as the understanding of the system and its parameters evolves.
- Interaction with resources occurs by using one of few predefined methods on them, such as the verbs defined by the HTTP specification. Applying an HTTP-verb to a resource results in a transaction of a *resource representation* that is defined by the media type or hypermedia type agreed upon by the service participants through a mechanism called *content negotiation* [32, section 12]. Typically, browsers ask for an HyperText Markup Language (HTML) representation of a resource which they then render as a web page for humans (as originally intended upon the invention of the Web). However, many other media types [32, section 8.3.1] exist. This enables supporting different needs, such as providing HTML for humans and JSON-representations for software [118, section 3.3.2].
- In REST-based services, all service interactions must be stateless: in other words, it must be possible to determine the results of a

request solely based on the information contained within the request message, resulting in an exchange of *self-descriptive messages*. *Stateless* here refers to the application state, in other words the progress and history of the interaction between service consumer and -provider. However, service interactions have an effect; this effect is stored in the resource state [112, page 11].

▼ [94, section 1.2]

- Last, it is an essential constraint of REST that the interaction between user and service is driven by the selection of choices provided in the resource representations [31], such as links and forms. This constraint is called HATEOAS, and humans make use of this principle successfully every day when browsing the Web to achieve their goals without first reading documentation on how to navigate a website.

▲

The success of the Web and its age show that REST is indeed "software design on the scale of decades: every detail is intended to promote software longevity and independent evolution" [31, comment 8].

Web APIs

Application programming interfaces (APIs)—as their name suggests—are both a specification for how software can use other software, and an offer to do so. They are one of two ways to use software at all; the other being user interfaces (UIs) for human users. APIs allow implementing high-level functionality by composition while hiding low-level details (encapsulation).

Software use through APIs can occur locally, on the same computer and operating system (OS), or remotely on a different computer connected via a network such as the internet. Different approaches to designing such remote APIs exist, resulting in different coupling characteristics [76]. When using HTTP to communicate over the network, remote APIs are called Web APIs [120, section 3.1].

▼ [94, section 1.2]

Web APIs are often *based on* REST such that they expose resources of which JSON-representations are transferred as reaction to HTTP requests, but do not fully implement the REST constraints. Specifically, instead of relying on HATEOAS, the possibilities to interact with a service are communicated through a static service interface description such as the OpenAPI Specification (OAS). Programmers then construct requests specific to a certain version of the API *at design-time*.

This is not RESTful because HATEOAS is an essential constraint in the sense that if it is not realized, a system cannot be RESTful [31, 118, p. 243 f.]. It also does not support loose coupling because *horizontal* interface orientation, a *shared* data model, *breaking* evolution, *static* code generation and *explicit* conversation are promoted, which indicate tight coupling [76].

APIs for software clients that are accessible over the internet and fully implement the REST constraints are called *hypermedia APIs* [117, section 3]. Frequently, REST-based HTTP-APIs are mislabelled as "RESTful" [80, section 5.2], therefore the term hypermedia API is used to denote APIs that realize all REST constraints, including HATEOAS.

▲

Most websites adhere to REST, and humans successfully use links and controls provided in HTML representations of resources to drive their application state every day. However, realizing HATEOAS for software clients is much harder to achieve. Humans are excellent at figuring out the meaning of controls and their potential role in achieving a goal based on natural language, context, and experience. Software clients are not; but they can process data at a much higher speed. This leads to the idea of the Semantic Web.

Websites can be seen as hypermedia APIs with resource representations tailored to humans; inversely, hypermedia APIs can be seen as websites for software agents.

2.3.3 The Semantic Web

The core idea of the Semantic Web is to be *explicit about the meaning of entities*, including links, in order to improve the accessibility of content to *intelligent software agents* [37]. Intelligent software agents are defined as "a computerised entity that: is able to reason (rational/cognitive), to make its own decisions independently (autonomous), to collaborate with other agents when necessary (social), to perceive the context in which it operates and react to it appropriately (reactive), and finally, to take action in order to achieve its goals (proactive)" by Cardoso and Ferrando [16]. Consequently, the idea of the Semantic Web raises the questions of how to make the meaning of data explicit; and how to enable the creation of intelligent software agents that use this data to achieve their goals.

In the 2001 article that presented the vision for the Semantic Web, it was made clear that the Semantic Web is meant to be a part of the Web; consequently, URIs are used to identify things and HTTP is used to transfer representations of resources.

"The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [10, p. 37]

However, HTML is not designed to encode semantics and facilitate processing by software, and thus not useful for resource representations in the Semantic Web. Instead, based on the fact that links relate sources to targets, RDF as a data model that relates subject nodes to object nodes via predicates, was developed. Subjects and predicates in RDF are URIs; objects are URIs or literals (ignoring blank nodes for simplicity). Serializations of RDF such as Turtle, TriG or JSON-based Serialization for Linked Data (JSON-LD) are used for representations of resources.

RDF and its serializations are the first building blocks of the Semantic Web from a technical perspective; they are complemented by RDFS and OWL for building ontologies; SPARQL for querying RDF graphs; and the (rarely adopted [42, p. 79]) Rule Interchange Format (RIF). All of these are developed and standardized by the World Wide Web Consortium (W3C).

With ontologies defined using RDFS and OWL, and RDF and its serialization formats to represent both ontological knowledge and specific data, the question of how to design interfaces to RDF datasets that facilitate the creation of intelligent agents remains. Options include SPARQL endpoints; Web APIs; Triple Pattern Fragments (TPFs) (discussed in section 5.2.1); Linked Data documents; and data dumps [120, section 3].

Linked Data documents contain triples in which an entity is either the subject or the object.

In this thesis, a hypermedia API that uses serializations of the RDF data model for its resource representations is used. It is a REST-compliant service *in* the Semantic Web. The expressivity of its interface lies between that of a SPARQL endpoint and a data dump; but, importantly (and in contrast to Linked Data documents and Triple Pattern Fragment (TPF)), it is not restricted to read-only access as all HTTP methods can theoretically be supported. Details on why this approach was chosen; on how software agents are enabled to choose between controls in the resource representations; and on how it was implemented are given in section 4.2, section 5.2.3 and section 5.3, respectively.

Review articles on the history and state of the Semantic Web detail application areas, research topics and -methodologies, and achievements of the Semantic Web community; also, it is noted that the Semantic Web has become an established field of research [37, 42]. With regard to future developments, Hitzler [42, page 83] states "one of the key quests is about finding out how to piece together contributions [...] in order to provide applicable solutions"; while Verborgh and Vander Sande [116, section 8] more specifically urge to re-focus research efforts on the Web aspect, in other words *distributed* semantics and "a lot of small data sets instead of a few large ones, and [...] heterogeneity instead of homogeneity".

Kirrane details her analysis of obstacles to the implementation of intelligent agents using the Semantic Web in [48], summarizing requirements and core architectural components as well as literature on Semantic Web agents and their potentials. While very interesting, the article is far beyond the scope of this thesis.

With regard to intelligent software agents, Kirrane and Decker [49] point out that even though intelligent agents always were part of the Semantic Web vision, there are still significant open research challenges from a data management perspective, from an application perspective and from a best practices perspective. The authors call for basing the development of intelligent agents on the FAIR principles since they see a "strong connection between said principles and Semantic Web technologies and Linked Data principles" [49, p.3].

2.4 Relevant Aspects of Software Engineering

The purpose of the software implemented as part of this thesis is—first and foremost—to demonstrate the feasibility of the ideas; but also to aid in understanding them. With regard to the expected quality, the software should be seen as a proof of concept. Nonetheless, it is a goal to align the implementation with the current state of technology and thereby ensure that, if the software were to be used in production systems, there are no design choices that contradict achieving expected characteristics.

Software that realizes today's expected non-functional characteristics for Web APIs is known as a cloud-native application (CNA); entailing that the software is provided *as a service*.

This section defines the core terms to describe service-orientation, an architectural style for structuring complex distributed applications in a way that is manageable and scalable; and introduces the key ideas followed to ensure that the developed M&S hypermedia API is a CNA.

▼ [94, section 3.1]

Service-oriented Architecture (SOA)

Service-oriented architecture is a "paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [75, line 128 f.]. Services are seen as "the mechanism by which needs and capabilities are brought together" [75, line 174]; in other words, a service describes the capability, the specification and an offer to perform work for someone. service-oriented architectures (SOAs) are seen as a way of structuring and offering functionality that promotes reuse, growth and interoperability [75, line 175] by focusing on tasks and business functions and acknowledging the existence of ownership boundaries.

The OASIS Reference Model for Service Oriented Architecture [75] is intended to provide a foundation for analysing and developing specific SOAs by giving definitions, explanations and examples of relevant aspects in a technology-independent manner. The reference model identifies six major concepts pertaining to services: visibility, service description, interaction, contracts and policies, real-world effect and execution context. Achieving a *real-world effect*, which can either be the retrieval of information or changes to the shared state (the knowledge that service provider and service consumer share), is the reason for using a service. Using a service means *interacting* with it through the service interface, typically by exchanging messages. The specifics of how to interact with a service are detailed in the *service description*. Interaction is only possible iff the service is visible to consumers. *Visibility* comprises awareness, willingness and reachability. Assuming that potential consumers are aware of the service's existence, reachability is defined by the *execution context* (the "set of infrastructure elements [...] that forms a path between those with needs and those with capabilities" [75, line 720 ff.]). Willingness to interact is governed by the *contracts* agreed upon by the service participants and/or the *policies* enforced by policy owners. ▲

From a more practical point of view, the *microservice* architectural style has been found helpful to implement scalable solutions. It is defined as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API" by Fowler and Lewis [35]. The authors further point out that "these services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies".

In this thesis, the terms service and microservice are used interchangeably, in the sense explained in detail in [35].

Central to (micro-)services is the definition of the service interface because it is the only means by which a consumer interacts with the service. The REST constraints can be seen as guiding constraints to create interfaces that support loose coupling, keeping longevity in mind; and SOA can be seen as a way for the service provider to structure applications such that they are scalable and maintainable.

▼ [93, section 1]

Cloud Computing

The term *cloud computing* denotes a set of desirable characteristics for accessing a set of computing resources over the internet, as well as characteristic service models and deployment models [62]. From a user's point of view, the essential characteristics are that software or computing resources are available *as a service* via the internet, meaning that the resources are readily available without the need for manual installation of hardware and/or software. Consumers can use services without the need for human activity on the side of the provider (*on-demand self-service*), often without apparent limitations, and they have access to metrics for their service usage (*measured service*). Users are billed according to service usage in terms of these metrics (*pays-as-you-go cost model*).

In alignment with the defining characteristics of cloud computing, Kratzke and Quint [52] propose the following definition for CNAs: “A CNA is a distributed, elastic and horizontal scalable system composed of (micro)services which isolates state in a minimum of stateful components. The application and each self-contained deployment unit of that application is designed according to cloud-focused design patterns and operated on a self-service elastic platform”.

DevOps

This definition of CNAs contains aspects pertaining to their *software development and -operations (DevOps)*: *cloud-focused design patterns*; *self-contained deployment units*; and *deployment on elastic platforms*.

DevOps is “a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability” [29]. Central ideas of DevOps are that the people who develop software, should also run it in production to facilitate software maintenance; and that testing, building and deploying software should be automated to facilitate frequent software updates (continuous integration/continuous deployment (CI/CD)). Core technical aspects/phases of DevOps include source code management; the build process; continuous integration; deployment automation; monitoring and logging; and knowledge sharing among relevant actors [29, table 4]. Details on the DevOps process for developing software as part of this thesis can be found in section 5.3.3.

A specific set of cloud-focused design patterns used in this work are known as the “twelve-factor app” methodology [123]. The twelve factors are best practices for providing Software as a Service (SaaS), synthesized from experience. Nine of the factors are about details of software development; whereas three factors concern themselves with build processes and -environments as well as admin processes. They are independent of any specific software or programming languages, but nonetheless very useful when implementing cloud services. Examples include using exactly one version-controlled codebase; explicitly declaring dependencies; storing configuration in the environment; and exposing services via port binding.

As self-contained deployment units, *containers*[1] are typically used. Container images bundle the application code and all of its dependencies; thereby, they are guaranteed to run on any computer which features a matching container engine, while being less resource-intensive than virtual machines, but still providing isolation from

[1] https://www.docker.com/resources/what-container/

other software running on the server, thus enabling better resource utilization.
Containers also form the basic building block for deploying applications on *clustered elastic platforms* such as Kubernetes[1], which allow scaling the available computing power beyond the capabilities of a single machine.

Containers can be seen as the common denominator for deploying software on cloud infrastructure.

[1] https://kubernetes.io/docs/concepts/overview/

3. Models, Simulations, and the Web

▼ [92, section 2]

Currently, factors that prevent a more widespread use of modelling and simulation amongst engineers include a lack of knowledge regarding concepts, modelling languages, algorithms and tools, the cost of tools and model libraries as well as problems with the usability of user interfaces and data formats. Also, current M&S environments are not designed to be used as part of a larger process. Factors preventing the general public from using modelling and simulation include missing awareness of its capabilities, missing background knowledge, missing feeling for the steps required to arrive at a solution and the challenge of using complex M&S software environments.

Modelling and Simulation as a Service (MSaaS) is an umbrella term for efforts attempting to address the aforementioned issues by combining the concepts and tools of SOA and cloud computing with the functionality of M&S. Thereby, it is hoped to help a wide audience make informed decisions when confronted with complex questions: the various methods, languages and tools summarized under the term modelling and simulation allow finding quantitative information about the behaviour of a system by numerically approximating the behaviour of a virtual representation of that system; SOA and cloud computing allow users to access this capability over the internet. ▲ Different acronyms are sometimes used to point out the focus of an implementation in the context of MSaaS; for example, Simulation as a Service (SIMaaS) is used to emphasize that the service is more about simulation than it is about models or modelling.

▼ [94, section 2.1]

Similar to the aforementioned idea of combining M&S with the Web, there have been attempts to combine ideas and tools resulting from research on the Semantic Web with those of M&S for almost as long as the vision of the Semantic Web exists. Many authors have focused on the use of ontologies for improving and supporting MBSE. Applications range from general process support aimed at better integrating knowledge from different sources, over working on the question of interoperability and composability of models, to model generation based on ontological system descriptions. Fewer work has been published on the use of hypermedia APIs in conjunction with M&S.

▲

In this chapter, the literature on Modelling and Simulation as a Service, the use of ontologies for MBSE, and the use of hypermedia APIs in conjunction with M&S is summarized to the extent it relates to the ideas investigated in this thesis. At the end of the chapter, a list of research gaps is formulated.

3.1 Modelling and Simulation as a Service (MSaaS)

▼ [92, section 2]

MSaaS is defined as a “means of delivering value to customers to enable or support modelling and simulation (M&S) user applications and capabilities as well as to provide associated data on demand without the ownership of specific costs and risks" by the specialist team MSG-131 at North Atlantic Treaty Organization (NATO) [70]. Cayirci [19] sees MSaaS as a "model for provisioning modelling and simulation (M&S) services on demand from a cloud service provider (CSP), which keeps the underlying infrastructure, platform and software requirements/details hidden from the users", which aligns well with the previous definition but highlights the use of cloud computing technologies. This aspect is important for distinguishing the current research activities from earlier attempts, summarized under the term Web Based Simulation (WBS), as retraced by Wang and Wainer [121, section 2]. Consequently, MSaaS is one form of SaaS [19, 58] that "inherits" the general benefits and challenges of cloud computing, but also offers opportunities peculiar to the field of M&S.

The proposed value of MSaaS [19, 70, section 2.6] lies in an increase in usability *and* functionality on the one hand and a possible decrease in costs on the other hand. Increased usability and functionality are expected to stem from better accessibility of M&S resources, their reuse for reproducing results and the creation of new functionality through composition of resources:

- The *accessibility* of M&S resources is increased through broad network access that allows on-demand self-service while having very light requirements on the client (hardware, OS, software).
- Accessible resources that hide their complexity and implementation from the user (encapsulation) make the *reuse* of knowledge possible, for example the provision of product models by companies.
- Also, providing an interface to an operational simulation environment can enhance the credibility of a simulation study by ensuring its *replayability* and makes sure that a user can always access the latest version of the software.
- Implementing new functionality by *composition* of services is facilitated by provisioning M&S resources as a service. Composition can reduce the time and effort needed for implementation and extend the scope of M&S resources by using them in conjunction with functionality that is not normally available in an M&S environment. Moreover, an increased value of M&S resources could stem from new use cases (UCs) and target audiences for M&S technology. One example of added value through composition of services is using M&S functionality from within online training and learning resources: by accessing M&S services in the background, learning resources are enriched by the possibility to interact with simulations [103].

A decrease in costs can be achieved by measuring the service usage, which allows pay-per-use options, by scaling the infrastructure according to demand and by increasing the degree of capacity utilization through resource pooling. Furthermore, the service consumer does not need to invest in hardware, software and personnel, which could increase the willingness of companies to try to integrate M&S into their workflow.

Researchers have worked on combining M&S with web technologies for more than 20 years by imagineering the possibilities [33, 58], proposing architectures for implementation [22, 128, 79, 89], pondering risk, trust and accountability [18] and investigating the composability of M&S services [106, 109]. In section 4 of the article "'Grand challenges for modeling and simulation: simulation everywhere—from cyberinfrastructure to clouds to citizens'" published in 2015, Tolk analyses the technical and conceptual requirements for cloud-based M&S from a high-level perspective. The technical requirements are seen as challenging, but manageable using approaches established in the context of systems engineering (SE) and software engineering [101, section 4.2]. In contrast, the *composability* (consistent representation of truth) of M&S services is seen as an open research problem which is unique to MSaaS (meaning that MSaaS is not just an instance of Software as a Service, but a distinct direction of research). Both methodological and conceptual metadata (compare section 2.2.2) are seen as required to enable composability; but overall, creating composable M&S services is put forward as an unsolved *grand challenge of M&S*.

▲

" To capture the conceptualization underlying the simulation system and make them accessible to ensure composable M&S services in a cloud is one of the grand challenges that needs to be addressed [...]" [101]

As for the technical challenges of realizing cloud-based M&S, three recent reviews on MSaaS outline the design space and identify high-level requirements and architectural choices that should guide the design and implementation of specific MSaaS solutions.

▼ [93, section 2]

First, Shahin, Babar and Chauhan map out the architecture design space (ADS) for MSaaS by presenting the results of a systematic literature review (SLR) performed with the aim to identify and describe the state of the art [88]. They categorize the primary studies considered according to different criteria such as the architectural style, the main drivers for architectural decisions, and quality attributes. Additionally, they ponder the implications of the chosen architectures and identify strengths and weaknesses. The authors conclude that MSaaS-realizations most often use a *layered approach* to build applications; that *containerization* is employed to improve deployability; and that *effective interfaces for end users* that hide complexity and technicalities motivate their development [88, section 5].

Second, Hannay et al. [40] reason about the infrastructure capabilities they deem necessary for realizing entire MSaaS ecosystems at scale. Based on "a systematization of concepts from ongoing deliberations on MSaaS" [40, section 3], the authors first review the service concepts of the NATO MSaaS reference architecture [39] and then elaborate on the functionality required for realizing MSaaS ecosystems. Their reasoning is structured around the themes *data management*, *service description and -discovery*, *composition and interoperability* and the *management of different components*. The findings are mostly conceptual in nature, useful for verbalizing and contextualizing design questions and -decisions when faced with implementing specific SOAs containing M&S capabilities. The authors also note that solutions for supporting, yet—from an operational perspective—highly relevant functionality such as logging, metering and monitoring are readily available.

Third, Kratzke and Siegfried [53] focus on the consequences expected from leveraging the cloud for M&S services. Using their work on and definition of CNAs [52] as a basis, they propose a definition for what cloud-native simulations (CNSs) are in terms of a textual definition [53, section 4.3], a cloud-native simulation stack and a cloud simulation maturity model. They summarize the software engineering trends in cloud computing as the evolution of deployment strategies to maximize resource utilization (smaller deployment units, elasticity); the use of microservices as an architectural style that supports the aforementioned; and the emergence of microservice engineering ecosystem components for container orchestration, monitoring, et cetera. The authors conclude that the same trends are to be expected for CNS architectures and that, like CNAs in general, CNSs should strive to isolate state in a minimum of stateful components.

▲

As stated in section 1.2, the work presented in this thesis focuses on models and simulations as understood in the context of the equation-based, object-oriented modelling language (EOOL) Modelica and the FMI standard. Consequently, previous work on providing MSaaS in the Modelica community is summarized next.

▼ [93, section 5.1]

Tiller [102] motivates the use of web technologies for the design of engineering tools in general, detailing potential benefits for non-expert users. The `FMQ` platform and its HTML5-based interface for human users are outlined; the use of a hypermedia API as the backend of the `FMQ` platform is hinted at, but no details are given. The `FMQ` platform also uses FMUs as an executable form of a single model to be simulated using a single solver. It is a proprietary product of Xogeny, Inc.

In their 2017 presentation of the `modelica.university` platform, Tiller and Winkler [103] motivate the high-level requirements for and architecture of the `Aperion` platform by Xogeny, Inc (presumably the successor of the `FMQ` platform) in addition to presenting the `modelica.university` website itself. With regard to concepts and the chosen technology stack, the `Aperion` platform seems to be very similar to the work presented in this paper, but it is a commercial product and specifics are consequently not available publicly. This also applies to the use of truly RESTful technologies such as hypermedia representations and generic software clients that use them.

Bittner, Oelsner and Neidhold [12] outline work on a web application based on FMI 1.0 for co-simulation. Compared at a high level, the application's architecture is similar to what is presented in section 5.3.1 as there is also at least a conceptual separation between API, storage, simulation components and front-end. The provided UI directly supports ranges for setting parameter values and seems to be intended for use by engineers. Details or source code are not readily available.

`FMIGo!`[1] is a set of software tools for executing several coupled FMUs over the internet. It is described in a paper by Lacoursière and Härdin [55] and available[2] under the MIT licence. In contrast to the concepts of the MSaaS-implementation explained in section 5.2 and the design concepts of the `FMQ` platform, `FMIGo!` choses not to make use of the design principles of the web (REST) and instead expose the capabilites of FMI withouth further abstraction through the use of (low-level) message passing protocols.

[1] `https://www.fmigo.net`
[2] `https://mimmi.math.umu.se/cosimulation/fmigo`

Lacking the simplification of seeing an FMU as nothing but an executable form of one model with one solver, `FMIGo!` allows/demands chosing master algorithms for co-simulation and exposes necessary numerical details; but this clearly makes it a specialized tool for simulation experts.

Elmqvist, Malmheden and Andreasson [28] present the Web Architecture for Modelling and Simulation (WAMS) in terms of several use cases, the corresponding web application aimed at engineers with training in M&S and a very brief overview of its software architecture. FMUs are used for simulation using `PyFMI` and it is claimed that a REST API is used for exposing capabilities of the server, but the extent to which the REST constraints are realized remains unclear. It appears that WAMS is an internal project of Modelon AB.

Since version 0.2.25 (released in November 2020), `FMPy` features the possibility to expose the UI of `FMPy` including the ability to simulate FMUs as a web application. Consequently, this approach falls in line with the WAMS web application, also intended to be used by engineers, and thus caters to use cases different to those that motivate the work presented in this thesis. ▲

3.2 Ontologies and Model-based Systems Engineering

Successful applications of ontologies in conjunction with M&S have been reported in three main categories: ▼ [94, section 2.1]

Support for Model-based Systems Engineering

There is data that is essential to the MBSE process, but not a model or simulation per se, such as requirements, changes, and configurations. This data is the focus of the OASIS Open Services for Lifecycle Collaboration (OSLC) specifications. OSLC aims to "enable integration of federated, shared information across tools that support different, but related domains" [72, sec. 2]. Technically, OSLC is based on the W3C recommendation Linked Data Platform (LDP) and consequently the exchange of RDF resource representations over HTTP. A core specification defining features of compliant interfaces is complemented by application-specific specifications; currently, the specifications for the query language used, requirements management and change management were published as OASIS standards[1]. There is no specification directly targeting M&S. In contrast to the work presented in this paper, which emphasizes support for generic software agents, OSLC has a strong focus on human end-users, as for example shown through the 'resource preview' [73] and 'delegated dialogues' [74] features.

El-khoury [47] reviews the adoption of OSLC in commercial software packages and summarizes the functionality as well as the envisioned consequences of the chosen software architecture from a practical perspective. The author concludes that the software architecture of OSLC allows for scalable, decentralized solutions in a heterogen-

[1] https://open-services.net/specifications

eous environment that changes with time by adhering to the REST constraints and using RDF as a data model that relies on interlinking entities and communicates their semantics without requiring adherence to a fixed schema of supported data fields [47, p. 25 f.]. Consequently, OSLC is seen as useful for tracing lifecycle information such as requirements across applications. The creation of the links that encode this trace, facilitated through delegated dialogues and resource preview, is identified as the most commonly implemented functionality [47, tbl. 7, p. 25].

König et al. [50] present a proof of concept-implementation that allows tracing virtual test results over simulation results and models back to the requirements which are evaluated through the virtual tests. The solution is based on OSLC and traceability information is sent from the different applications used to an OSLC server (denoted as daemon) via HTTP, but all applications including the daemon run locally only. The traceability information is mostly created automatically and stored in RDF in a graph database against which queries in the database-specific query language can be evaluated. Mechanisms for including traceability information provided by others are provided during startup of an instance running locally. Furthermore, some information can be extracted from git history for tools which store their state in textual form. It is concluded that the approach is well-suited for projects that require documentation of links between MBSE artefacts, as for example in safety-critical applications [50, p. 176].

Interoperability and Composability

Hofmann et al. anticipate that "for many technical domains and artificial systems, ontologies will be able to ensure the interoperability of simulation components developed for a similar purpose under a consensual point of view of the world" [44, p. 142], but point out that difficulties are expected for non-technical systems. Axelsson [5] relates each of the LCIM levels to the Semantic Web technology stack with a special focus on the RDF data model, gives specific examples and also concludes that RDF is suited to resolve interoperability problems.

The use of ontologies to improve the MBSE process with a focus on enforcing consistent views on a product among its developers is investigated in detail by Tudorache [110]. The work is based on the observation that the different syntaxes involved; the different views on a product and its semantics; and the lack of formal model transformations between different modelling formalisms lead to a risk for inconsistencies and misunderstandings, and makes tracing changes as well as the algorithmic, combined use of models in several formalisms difficult. Tudorache provides a formal definition of 'consistency'; defines ontologies that enable encoding different views on a system based on high-level patterns in system design (part-whole relations, connections, constraints, ...); and provides a framework for consolidating viewpoints as well as an algorithm that evaluates their consistency. It is concluded that the use of ontologies can lead to higher quality models and a better MBSE process. However, challenges are expected when introducing the use of ontologies at scale.

Ontology-driven Modelling
denotes the idea of first using referential ontologies to describe the logical structure and component functionality of a system and then inferring the simulation topology as a composition of component models via reasoning. For this, domain concepts are mapped to their representation in a model, which are described using methodological ontologies.
For example, Mitterhofer et al. [64] create a system model from a system description that encodes project-specific information using an appropriate ontology in the context of building performance simulation (BPS). This is enabled by annotating the component models with model-specific and domain-specific information and then using a reasoner to infer connections between models. Wiens et al. present similar work for creating digital twins of wind turbines as an example of large, modular multi-domain systems [122]. Both base their implementations on FMI as the format for the component models and the System Structure and Parameterization (SSP) standard [67] for the specification of the topology, in other words the connections between the FMUs. Neither details how the KGs used are populated and to which extent the triples are derived automatically; and both describe a local, non-distributed process. The approach is seen as promising in both publications.

However, there are limitations to the usefulness of ontologies in general. First, Hofmann, Palii and Mihelcic [44, pp. 139–141] point out that any language is insufficient for representing reality, and that the meaning of relations cannot always be grounded in logic. Second, the descriptive and normative nature of ontologies need to be balanced, which is expected to be especially difficult for non-technical systems [44, p. 144 f.]. Third, the value of using ontologies depends in part on their adoption in the M&S community—the more ontologies are used, reused and interlinked, the more useful they can become [108, p. 134]. ▲

3.3 Hypermedia APIs and M&S

As for the use of hypermedia APIs for exposing, querying and using M&S capabilities, only a few lines of work were found. ▼ [94, section 2.2]

First, Bell et al. [8] motivate the use of a methodological ontology combined with referential ontologies to discover and retrieve models from distributed sources for *local* aggregation and simulation in standard simulation environments. They summarize their reasoning and implementation process using the discrete-event-based simulation of a supply chain as an example. A KG is built—using the Discrete-event Modeling Ontology (DeMO) [90] as the methodological ontology and an application-specific referential ontology—which is then used for answering instance retrieval queries in a way that both exact matches *and*, through reasoning, possible alternatives are returned. The results are links to models which can then be downloaded for inspection or use in a local simulation. The developed framework consists of several services, but is ultimately used by humans; generic software agents are only mentioned in the "related work"-section.

Second, Tiller and Winkler outline the motivation for and use of a hypermedia API to build a framework acting as a "content-management system for scientific and engineering content" [103]. However, details about the implementation, source code or insight into the observed benefits and/or drawbacks of using a hypermedia API over a plain Web API are not available publicly.

3.4 Research Gap

In chapter 2 and the previous sections of this chapter, both the broad concepts and specific publications related to this work were surveyed to properly ground this thesis within previous work in the scientific community. Below, the research gaps identified while reviewing relevant publications are stated.

- **Semantic Web concepts and -technologies were mostly combined with M&S with the focus on using ontologies to describe models, not on providing models and simulations as part of the Semantic Web.**
- **Only a few of the papers found discussed using ontologies for improving the findability of models in a distributed setting.**
- **It is not clear from the publications on ontology-driven modelling using FMUs [64, 122] in how far the representations of FMUs in RDF are generated automatically.** No open-source software or ontologies were published and only local, not distributed, processes were described.
- **No open-source parsers or ontologies for creating representations of FMUs in RDF were found.**

- **No explicit consideration of the FAIRness for M&S entities and capabilities in the context of dynamic system simulation using Modelica and/or FMI was found.** The goals of providing MSaaS are in fact similar to those of the FAIR principles, but to the best of our knowledge, the principles have not been applied to M&S resources and capabilities in the Modelica/FMI community.
- **No work that explicitly makes loose coupling a design goal for providing MSaaS was found.** The exception is [103], but the paper only hints at this aspect and discusses the developed solution from a practical, less conceptual perspective.
- **No discussion of the consequences of fully realizing the REST constraints for providing MSaaS was found.** Bell et al. [8] and Tiller and Winkler [103] describe applications that involve hypermedia APIs for accessing M&S entities. However, advantages or disadvantages of

their use are not discussed in detail and no open-source software is available.

- **No studies on the use of generic agents for achieving tasks involving M&S were found.**
- **No applications of the Pragmatic Proof Algorithm (PPA) beyond the examples discussed in the publications presenting it were found.**
- **The original version of the PPA is unable to reliably account for requirements on requests that only become known at run-time.**

To summarize, there are promising results on using ontologies for MBSE and on exposing M&S resources and capabilities as a service, but the potential of using hypermedia APIs for exposing M&S functionality in a FAIR and loosely coupled manner has not been explored in detail yet.

4. Thesis Concept and -Hypotheses

In general, the resources (source code, data) and capabilities (solver, software, expertise) required to solve tasks in a MBSE context are distributed both with respect to their ownership by organizations or organizational units and their physical location. For example, resources could be models, parameter sets, input data, simulation results, system properties, and data sets for evaluation as well as requirements for and changes to the task at hand. Activities involving the use of these resources include the instantiation and simulation of a model, querying for specific information or retracing the role of resources and activities in the overall MBSE process. Model, parameter set and the ability to run simulations could be owned by a supplier of components, whereas another company is using the results of simulations of the component model with their own input data in a model-based process to evaluate the fitness of a design for a given purpose.

For a specific instance of such a problem, it would be a straightforward, in other words a matter of "just programming", to implement some mechanism that allows the stakeholders to share the necessary resources and activities. However, an application-specific solution is unlikely to have a value beyond its original purpose.

Therefore, it is aimed at providing a *reusable building block for SOAs* that acknowledges that the consumers of a service and their intentions are in general *unknown*. The concept for this building block is to address core issues that hinder reusability of M&S resources and capabilities and to aim for desirable characteristics that support it, which are presented in the following subsection. Next, approaches that might be suitable, but have not been explored in detail yet, are identified. Then, hypotheses for the resulting characteristics of their combined application to providing M&S capabilities in a distributed system are formulated.

4.1 Issues and Desirables

Based on my understanding of how models are created and used to solve tasks, I see three issues that unnecessarily complicate or even prevent their reuse—both the intentional and the serendipitous reuse—and thus limit their value: knowledge that is inaccessible; a lack of Findability, Accessibility, Interoperability, Reusability (FAIRness) and machine-actionability; and the heterogeneity of interfaces when exposed *as a service*.

Lack of FAIRness

Even if models are interoperable because they are implemented in the same modelling language, *finding*, *accessing* and *reusing* them is challenging in practice, even for humans.

For example, searching for models to reuse is typically a manual, time-consuming process. When using Modelica, models are organized in libraries. There is a search engine[1] [104] and a manually maintained list of libraries[2] that can be used for searching libraries which might contain a certain model, but the search is limited to the exact words used in the title and description of the library (for example, searching for "PV" returns two libraries, whereas searching for "photovoltaic" returns five). Then, one needs to manually go through the libraries and look for models based on the model name and its documentation. In other words, searching for models is based on their syntax and implicit or non-structured data. Searching for models based on their semantics would facilitate the process. The current best practice for accessing models written in Modelica is that they are hosted in a git-repository which is then manually installed by cloning. Reuse is often facilitated by supplying a licence for the repository.

Interface Heterogeneity

Heterogeneous interfaces that are not self-explanatory for software agents may not be a problem per se, but it means that programmers are required to create or change client applications every time there is a change in the service interface or a new service interface needs to be used. Therefore, using many interfaces could become prohibitively expensive.

Models and simulations exposed through a non-self-explanatory API essentially result in a *different interface for every model, model instance and simulation* because each model exposes different parameters, inputs, outputs, and other settings.

It might be possible to find a very abstract interface that is the same for every model; but this would likely require more external documentation and complicate the implementation of the API, among other reasons because implementing input validation becomes laborious. Therefore, an interface that is too abstract is undesirable.

Inaccessible Knowledge

Knowledge about models that is not implemented as the model equations themselves is typically only accessible to a certain degree (usually lower rather than higher). Reasons for this include that documentation is only available in natural language; that the documentation is not directly part of the model and can therefore get separated from it; that it is only documented partially; et cetera.

Examples for the type of knowledge that often only exists in the minds of those who originally created the model include knowledge about the model's experimental frame, in other words the conditions for which a model returns valid results; knowledge about the achieved accuracy; about data and results used for validation; the origin of model equations, default values and lookup tables; successful or unsuccessful uses of the model; how to test for regression; and more.

[1] https://impact.github.io/
[2] https://modelica.org/libraries/

In addition to these core issues, I see desirable functional and non-functional characteristics that a solution providing access to M&S resources and capabilities in a distributed setting *should* exhibit.

Querying
Having the ability to query available resources on a semantic level, in other words by the meaning and interrelations of entities, directly improves findability and traceability.

Reasoning
The ability to reason about available data has several advantages:

- it allows filling knowledge gaps—this is crucial because it does not make sense to *explicitly* add every possible triple when creating an RDF representation of something;
- it improves querying because more query formulations lead to results; and
- it adds functionality such as determining whether two models are composable.

Traceability
The ability to reliably and explicitly connect M&S entities to requirements; changes; and provenance data as well as the ability to retrace the origin of some piece of data are useful for creating reliable systems in general. Moreover, building trust into information found is facilitated by enabling subsequent querying for provenance information describing this information. Therefore, traceability is often required in safety-critical applications.

Declarative Problem Formulation
A declarative problem formulation, in other words the ability to describe the task to be solved in a way that is independent of *how* it is solved, is desirable. The reasons for this are that such separation between problem formulation and the applied algorithm accelerates finding solutions and eliminates a source of errors (users do not need to know anything about the specifics of a solver). It also allows experimenting with different solvers (some solvers might perform better than others).

Loose Coupling
The devised software is intended to be used in a distributed setting without a centralized authority that can enforce an interface to be used. Moreover, it is intended to be used by consumers that are unknown at design-time for purposes that are also unknown. Therefore, it is desirable that the software supports its use in loosely coupled systems of systems (compare section 2.3.1).

State of the Art
From a non-functional perspective, the implementation should represent the state-of-the-art for services made available via the internet. This includes the defining characteristics of cloud computing (on-demand self-service, horizontal scalability, ...; see section 2.4) and the use of a modern, adequate technology stack.

4.2 High-level Design Choices

Seen from an abstract perspective, it is the hypothesis of this thesis that a *specific set of high-level design choices* is suitable to address the issues outlined in the previous section, such that the desired characteristics can be realized. Furthermore, it is hypothesized that the approach has not been investigated in detail before and that the resulting software can be useful in specific situations.

▾ [94, section 3.1]

The goal of this work is to provide M&S capabilities in a way that the FAIRness and actionability by generic software agents improve and that loose coupling is supported (section 1.2). The FAIR-principle A1 demands that "(Meta)data are retrievable by their identifier using a standardized communications protocol". Therefore, the first design choice made is to provide M&S capabilities *as a service*, meaning that the capabilities are intended to be used within a SOA. Alternatives to a SOA, such as spawning local instances of simulation environments for each user or exposing technical interfaces remotely (as opposed to more abstract interfaces that directly provide business value), were disregarded because they contradict the ideas and goals of this work.

The second design choice made is to design the service for the Semantic Web as the intended execution context. The reasons for this are that the FAIR-principle I1 demands the use of a "formal, accessible, shared, and broadly applicable language for knowledge representation", as well as the overall similarity of the FAIR-principles and Semantic Web concepts and technologies [49, p. 3]. An alternative would have been to develop the service for use within a custom platform/ecosystem that demands adherence to a centralized definition of interfaces and semantics. However, this would have contradicted the goal of achieving loose coupling; might have prevented the reuse of concepts, technologies and software components developed in the Semantic Web community; and limited the usefulness of the developed solution to said custom platform. The choice for the Semantic Web as the intended execution context entails the use of specific technologies to connect service instances and consumers: first, the access to the service via the internet and the corresponding protocol stack; second, the use of HTTP, URLs and hypermedia as core mechanisms of the Web; third, the use of a graph data model in conjunction with a corresponding schema language and query language to represent all (meta)data, restrictions on and subsets of it; and fourth, the use of ontologies based on DL to represent knowledge. The recommendations for specific technologies by the W3C are followed, meaning that RDF, RDFS, Shapes Constraint Language (SHACL) [51], SPARQL, and OWL are used.

The service interface is the only means through which consumers can interact with a service. Therefore, the service interface defines the level of abstraction at which consumer and provider interact; it defines what interaction means (for example, the exchange of RDF-serializations using HTTP); what requirements must be fulfilled by consumers; and, importantly, the characteristics of the service with respect to coupling. For this work, loose coupling and the service's use by generic software agents are desired. Realizing the architectural style of the web, REST, can result in loosely coupled systems [76, p. 919]. Therefore, it was decided that the service interface should be realized as a hypermedia API, in other words an interface that fully realizes the REST constraints and uses serializations for resources that are machine-actionable. Alternatives would have been to *base* the interface on REST, but not realize the HATEOAS-constraint and rely on a static service interface description instead; or to realize a Simple Object Access Protocol (SOAP) or remote procedure call (RPC)-style interface. However, neither of these supports loose coupling ([76, p. 919], also compare figure 7.1).

Having decided on realizing a hypermedia API intended to be used in the Semantic Web as the execution context, the questions "which consequences do these choices entail for the representations of resources?" and "how to realize the HATEOAS constraint?" arise. Both HATEOAS and the exchange of self-descriptive messages required by REST imply that the service description must be included in the resource representations transferred as the reaction to HTTP requests by consumers. This means that, in addition to the data itself, resource representations should also contain metadata, context and controls [113]. *Metadata* can be about the triples that represent the resource exposed, as well as about the resource *representation*. It contributes to answering the question "what is this resource?". *Context* is created by providing qualified references to the resource itself and to other resources; it answers the questions "where am I?" and "what else may be interesting?". *Controls* provide answers to the questions "what can I do with this resource?" and "where can I go from here?". They are actionable and provide specific information on how to construct executable requests; thereby enabling the HATEOAS principle.

REST-based HTTP-APIs typically exclusively provide data in their resource representations, but software agents need—and thus should have access to—metadata, context, and controls even more than humans browsing the Web because they are far worse at interpreting contextual clues or relying on experience with similar websites, as humans do. However, if triples that encode metadata about the resource representation, context or controls were included in the same graph as the data triples, the use of the RDF graph by clients would be unnecessarily complicated since clients likely would want to separate the different parts, for example for counting how many items there are in a collection [113]. This problem is avoided if the data is put in the default graph and the other parts in dedicated separate graphs, which mandates the use of an RDF serialization that supports quads. An example will be discussed in section 5.2.2.

To summarize, the decisions to provide functionality *as a service* within the Semantic Web through a hypermedia API mean that consumers interact with the service by exchanging *resource representations* in serializations of the RDF data model that support named graphs via HTTP messages. These messages are independent of any possible prior messages, in other words self-descriptive, and they contain metadata, context and controls in addition to the actual data in order to support the HATEOAS principle and in order to facilitate the service's composition and execution by generic software agents. For this work, it is assumed that consumers are aware of the service's existence and that the service participants are willing to interact with each other without restrictions: in other words, service discovery as well as the negotiation of contracts or the enforcement of policies are out of scope because they are irrelevant to the research questions.

▲

4.3 Hypotheses

Compare "*Is This Really Science? The Semantic Webber's Guide to Evaluating Research Contributions*" by Bernstein and Noy [11, sections 2.5, 3.2, 3.3 and 3.4].

Based on the issues, desirables and research gaps as well as the high-level design choices, a set of hypotheses were formulated that define the scope of the research presented. There are two types of hypotheses: the four main hypotheses H1–H4 try to establish the *relevance* of the pursuit. In other words, they address the questions "so what?" and "who cares?", *independent* of how the problem is solved. In contrast, the hypotheses on the second level are intended to establish the technical merit of the proposed solution, meaning that they represent aspects that are only relevant to those who care about *how* the result is achieved. Because the corresponding implementations are made available openly, these second-level hypotheses can also be seen as technical contributions.

The hypotheses are amended by outlines of how they are intended to be verified or falsified as well as threats to their validity in order to better define their scope and to critically review their quality.

H1. **Representing M&S entities in RDF using the newly proposed and established ontologies enables advanced querying that was not possible using open-source components before.**
To verify this hypothesis, an exemplary KG is built containing triples about models and their context. Then, knowledge queries (queries about entities and their relations to other entities) are answered both with and without deriving additional data by means of reasoning about the explicitly added triples.
The validity of this approach is threatened by the possibility that there are other approaches to execute such queries; or that the queries chosen as examples are irrelevant in practice.

H1.1. **Researchers and software engineers can use the FMI-ontology and the `fmi2rdf` software package to represent FMUs in RDF, for which there was no open-source solution before.**
This hypothesis is not validated thoroughly, as for example by means of a survey among users. Instead, the publication of ontology and software under an open-source licence is seen as validation for enabling others to use it. The use of both as part of the M&S hypermedia API proves their functionality.
Due to testing the developed software with a limited set of FMUs only, it is possible that faults that prevent the software from working reliably for *any* valid FMU remain undetected.

H2. **The use of a hypermedia API to expose M&S entities and -capabilities increases their FAIRness compared to the current standard for exchanging dynamic system models.**

This hypothesis is evaluated by classifying the degree of realization for each of the 15 principles for both a model as a FMU and as exposed through the M&S hypermedia API. If the latter achieves a higher degree of FAIRness, the hypothesis is seen as validated.
However, the validity of this approach can be questioned for several reasons: first, there are usually more than one requirement per FAIR principle. Second, the principles' meanings are not always entirely clear. Third, the classification for a given principle might depend on implementation details, the modelling of resources exposed by the API and/or data supplied (or omitted) by users.
Moreover, the evaluation is only done by me, based on the FMI specification and the characteristics of the developed software. It is consequently unknown whether other researchers would arrive at identical classification results.

H2.1. **M&S engineers can use the implemented service to enable non-specialist users to generate simulation results using user-supplied data; and to provide immutable references to their models, model instances and simulation results.**
It is further anticipated that...
- due to its scope and publication under an open-source licence, the service fills a gap in the landscape of tools for the distributed simulation of FMUs; and
- the implementation exhibits the defining characteristics of cloud computing and matches the state of the art from a DevOps perspective.

This hypothesis is evaluated qualitatively by observing the characteristics of the implemented software, which is made available openly.

H2.2. **A M&S hypermedia API supports its use in loosely coupled systems.**
For evaluation of this hypothesis, the characteristics of the M&S hypermedia API for each of the coupling facets identified by Pautasso and Wilde [76] are determined and compared to its REST-based (but not RESTful) predecessor.
Compared to the evaluation of the FAIRness, the categorization is less subjective, but there is still room for interpretation as well as a dependence on specific implementation details.

H2.3. **The machine-actionability of M&S entities and -capabilities improves compared to the current best practice.**
Here, an increased FAIRness and the successful use of the API by the PPA are seen as observations that validate the hypothesis.
No generic software agents other than the PPA are tested.

H3. **In combination, the M&S hypermedia API and the extended PPA decrease the programming effort necessary to use M&S as a service compared to the current best practice.**
For validation, the simulation of two different models with different parameters and/or inputs are given as goals to an implementation

of the PPA. If it correctly determines and executes the necessary requests *without any code specific to the chosen API or models*, then the hypothesis is seen as validated because the current best practice, a REST-based HTTP-API, would require programming for each model used. Note that for this test, it is assumed that input data is readily available in the correct format.

H3.1. **Researchers and software engineers can use the implementation of the PPA to declaratively solve hypermedia API composition problems when the hypermedia APIs provides RESTdesc descriptions to communicate possible state transitions and the resulting public changes to the shared state. No open-source implementation of the PPA was available before.**
Like the ontologies and the `fmi2rdf`-parser, there is no survey or similar to empirically verify these claims. Instead, the implementation's use for the applications presented in section 6.2.2 is seen as validation for their proper functionality and its release under an open-source licence for the possible use by researchers and software engineers.
Threats to the validity of this hypothesis are that the PPA-implementation was only tested against two different hypermedia APIs and that not finding an open-source implementation resulted in claiming there was none (instead of, for example, contacting the authors to verify).

H3.2. **The proposed extension to the PPA allows properly handling request bodies which have requirements that are only communicated at run-time, which was not possible before.**
An example for requirements on request bodies that only become known at run-time would be setting the parameters for a model just added to an instance of the M&S hypermedia API by a user. More specifically, consider a model containing a mass: the requirements would for example communicate that its weight needs to be set; that the value's unit is kg; and that negative values are not allowed.
Consequently, this information is essential for constructing valid and complete requests to the API. In the original publications of the PPA [112, 119], this was not accounted for. Without modification of the PPA, it would thus be impossible to reliably construct valid and complete requests.
The proposed extension to the PPA to address this issue is presented in section 5.4.2.

H4. **A cloud-native implementation of the M&S hypermedia API combined with the extended PPA-implementation and a KG about M&S entities and their context improves the quality and/or speed of solution for realistic MBSE UCs.**
Several UCs were devised and implemented in order to provide evidence for the validity of this thesis; they are presented in chapter 6. Specifically, they intend to show that the approach...

  - enables finding models by their meaning and interrelations instead of by keyword only;
  - enables the parallel execution of independent simulations in the cloud;
  - allows solving declaratively formulated MBSE tasks;
  - allows combining knowledge about models and model instances with their broader context; and
  - can facilitate trust in the results obtained using M&S entities and -processes.

  There is a possibility that the UCs might be better solved using a different approach; that they are not representative of typical tasks in a MBSE context; or that engineers/programmers do not care about improved quality and/or speed because neither are relevant issues in practice.

For all hypotheses, there is a risk that relevant literature was not found and thus excluded from the literature review. Furthermore, there is a risk that even though the implemented software is intended to be independent of the UCs (or be generalizable in a straightforward manner), there are aspects specific to the UCs (or missing whatsoever) which have not been discovered due to limited testing.

The overall approach, namely deriving the hypotheses from perceived issues and desirables as well as the identified research gap, is based on assumptions. The first assumption is that the lack of FAIRness and machine-actionability of M&S entities and resources is indeed a problem worth solving. Second, it is assumed that the increased use of the (Semantic) Web by software agents performing tasks to support human users indeed allows realizing unfulfilled potential. Third, it is assumed that there are uses of M&S *when exposed through a M&S hypermedia API* that we don't even know of yet; meaning that is valuable to aim for creating a reusable building block supporting loose coupling instead of implementing a one-off solution tailored to a specific application. This further implies that it is valuable to think about relevant best practices; uses in conjunction with other services; and real-world UCs.

For all three assumptions, there are no references given. Instead, they are based on the author's understanding of the field and profession, which is limited and may or may not accurately reflect reality or omit important aspects. This possibility further threatens the soundness of the approach in addition to the threats to validity for the individual hypotheses stated above. Nonetheless, to the best of the author's knowledge, the assumptions have a good chance of holding up against the reality of MBSE practices.

4.4 Conceptual Overview

The design choices made to arrive at a service that provides both the desired functionality (access to system models and their simulation, semantic search; both accessible to generic software agents) and the desired characteristics (FAIRness, loose coupling) are summarized in table 4.1.

▼ [94, section 3.3]

Functionality, High-Level Aspect	Concept for Realization	Possible Alternatives with Respect to Choice	Resulting (or Chosen) Technologies
Software as a Service; service-oriented architecture	microservices	local instances simulation environment for each user; remote access technical interface	Node.js with Express.js for API; Python with FMPy for worker; Celery with RabbitMQ, Redis for queue
intended execution context	**Semantic Web**	custom/proprietary platform	HTTP(S), URLs, hypermedia; explicit semantics; graph data model
knowledge representation	**graph data model + schema language; DL-based ontologies**	*consequence of choice for Semantic Web*	RDF, RDFS, SPARQL, SHACL, OWL
service interface concept	**hypermedia API** → client-server constraints, uniform interface constraints: identification resources, manipulation through resources, self-descriptive messages, HATEOAS	REST-based HTTP-API + OAS; SOAP; RPC	—
resource representations	**self-descriptive representations containing data and explicitly separated metadata, context, controls**	data, metadata, context, controls in *same* graph	—
service interface functionality	**resource representations in RDF-serializations transferred to consumer as response to HTTP-request**	*consequence of choice for hypermedia API in Semantic Web*	RDF-serializations supporting named graphs, e.g. TriG, JSON-LD, …
semantic search	**Triple Pattern Fragments**	SPARQL server; no query interface at all	Linked Data Fragments Server.js
exposed resources	**immutable models, instances, simulations, results**; models persistent, others subject to TTL/LRU caches	more technical interface (FMU, …); all resources persistent	—
supported model type	**complete dynamic system models as causal MIMO-block; parameters can be set**	incomplete (component-) models; acausal models	FMI 2.0 for co-simulation as executable
contracts/policies; awareness; willingness	*out of scope*	contract negotiation /policy enforcement through additional service(s) or manual implementation	—
service composition/-execution by generic software agent	**Pragmatic Proof Algorithm + extension**	*out of scope*	own implementation in Python using EYE, requests, rdflib
enable planning	rules communicate state transitions and public changes to shared state	*consequence of choice for PPA*	**RESTdesc descriptions** (N3 rules)
non-functional characteristics	**cloud-native application** → on-demand self-service, measured, pay-as-you-go, horizontal scalability, …	disregard expected /proven characteristics and corresponding best practices	12factor-app; containerization; clustered elastic platform; separation API/worker through queue; …

Table 4.1: Summary of the design aspects, chosen concepts, alternatives and chosen technologies for implementation. Design choices are set in **bold**; the em-dash — is used to denote that something is not applicable. The chosen technologies listed in the last column will be discussed in section 5.3.

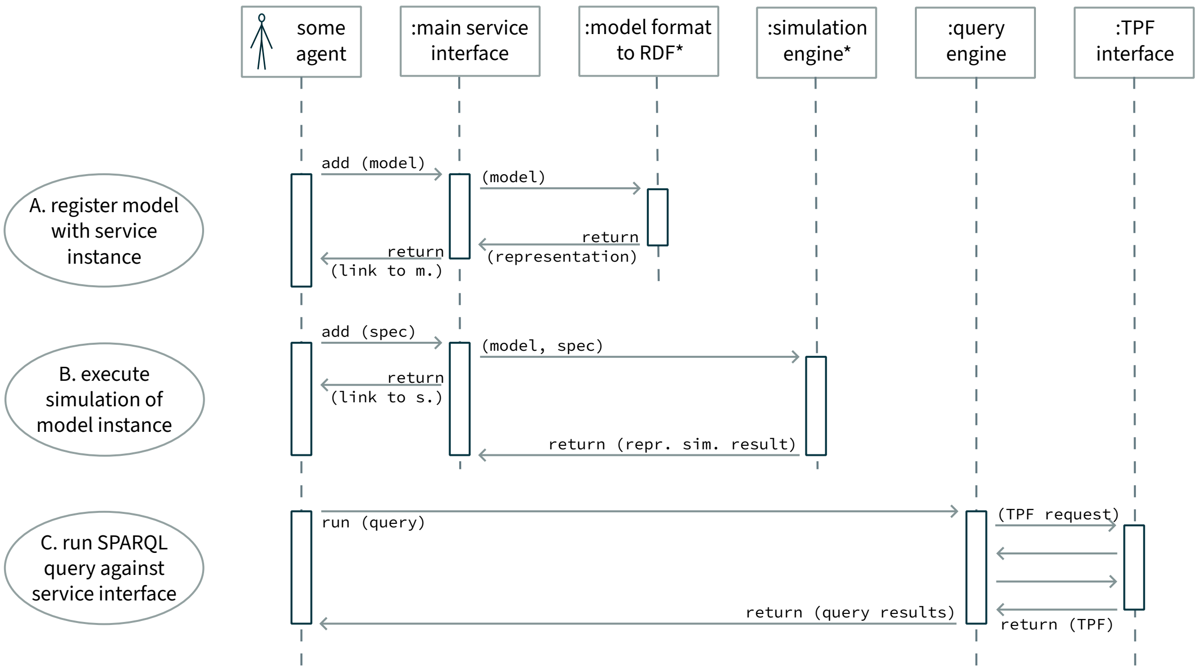

Figure 4.1: High-level sequence diagram for three main use cases of the service. Objects marked with an asterisk are specific to a certain model format such as FMI 2.0 for co-simulation.

To summarize how the design choices translate to the implemented software system, figure 4.1 visualizes three exemplary interactions between consumer and provider as a Unified Modeling Language (UML) sequence diagram. The objects and corresponding swim lanes refer to high-level parts of the software and not to specific technologies, therefore the diagram is intended to serve as a technology-neutral system overview. The first use case depicted (A) is the addition of a model to an instance of the service: the agent interacts with the main service interface to add the model, for example a FMU; in the background, an RDF representation of the model is generated and then integrated in the representation of the newly generated resource. The translation of the supported model format to RDF is dependent on the model format and therefore marked with an asterisk; the service interface is independent of the model format. The second use case (B) is the simulation of a specific model instance with specific initial conditions and inputs. Again, the agent sends the specification of the simulation to the main service interface. Specification and model are then passed on to the simulation engine which calculates the result. Third, the agent runs a SPARQL query in use case C. Since the service does not support SPARQL directly, the query is decomposed into a series of TPF requests by a query engine (not part of the service), which are sent to the service's TPF interface. The answers to these TPF requests are then combined to form the result of the original SPARQL query, which is sent back to the agent. ▲

5. Software Design and -Realization

The goals for the development of the M&S hypermedia API are to provide models and model instances over the internet; to allow the simulation of instances and provide access to the simulation results; as well as to enable querying the data held by a service instance. As motivated in the preceding chapter, the core design choices are to fully implement the REST constraints and to realize the characteristics expected by CNAs. The semantics of entities are to be made explicit by using the Semantic Web technology stack with the aim of showing the potential of the outlined approach to increase the FAIRness of M&S capabilities and to enable their use by generic software agents.

Seen from a high-level perspective, the tasks to be solved by a M&S hypermedia API are...

- to represent the capabilities of the application domain in terms of resources and applicable methods;
- to communicate these capabilities as well as the specifics of interactions to possible clients; and
- to build representations of resources in the supported formats.

The formats chosen for serializing resource representations define which information required by consumers can be communicated within the representations themselves and which aspects need to be communicated through additional documentation.

To demonstrate the advantages of an API that realizes all REST constraints and is built using the Semantic Web technology stack, two versions of the service exist. The first one is *based on* REST, but does not implement the HATEOAS constraint. This version is consequently denoted “REST-based HTTP-API”; it represents what is usually done when developing web applications and serves as the baseline for evaluating the advantages of the second version by comparison. The second version realizes all REST constraints, is thus denoted “hypermedia API”, and uses the RDF data model.

5.1 The FMI- and SMS-Ontologies

As a prerequisite for implementing the M&S hypermedia API, the ability to represent the entities of the application domain in RDF is required. Specifically, it is necessary to represent concepts of the modelling formalism used, such as FMI, in RDF; as well as to name (parts of) systems, models and simulations on a more abstract level, independent of the modelling formalism, and relate them to each other. Moreover, it is desirable to enable inferring more abstract representation from more specific ones via reasoning, in other words to encode data in terms of models and simulations from data about FMUs. For example, it should be possible to infer that a FMU is a `sms:Model` because all instances of type `fmi:FMU` are also of type `sms:Model`. Last, the representations should be derived automatically as far as possible, for example using a parser that reads the model format and outputs an RDF representation, also using a reasoner to infer additional triples.

5.1.1 Ontologies

▾ [94, section 4.1]

Three ontologies were developed using the Protégé editor [71]: the FMI-ontology allows describing FMUs in RDF; the Systems, Models, Simulations (SMS)-ontology allows relating (parts of) systems to (parts of) models; and the SMS-FMI-ontology captures the interrelations of concepts defined in the individual ontologies in order to enable a reasoner to infer triples using the SMS-ontology from triples about FMUs.

None of the developed ontologies contain individuals, in other words assertions about specific entities. These are intended to be created as part of a KG or as part of an application such as an instance of the M&S hypermedia API. A fictional excerpt of such a KG is shown in figure 5.1 with the intent to visualize the main concepts and roles of the FMI- and SMS-ontologies.

First and foremost, the ontologies are intended to allow unambiguously naming relevant entities and relationships in the context of the work presented in this thesis, in other words they are intended to serve as a vocabulary. So far, the ontologies are *not* developed and tested for the purpose of supporting ontology-driven modelling or drawing detailed conclusions about them via reasoning. Therefore, only little semantics is defined in terms of concept hierarchies or complex OWL statements at this point in time.

FMI-Ontology

The FMI-ontology is essentially a transcription of definitions in the FMI standard document [65] to RDF and OWL. Only minimal relations between concepts and roles are defined and the `rdfs:comment`-annotations are mostly verbatim copies from the standard.

Despite the simplicity of the FMI-ontology, it allows declaring FMUs and their variables including inputs, outputs, and parameters; as well as specifying their type and unit. Moreover, constraints on variables such as minimal, maximal or nominal values and the (limited) metadata specified in the FMI standard can be expressed.

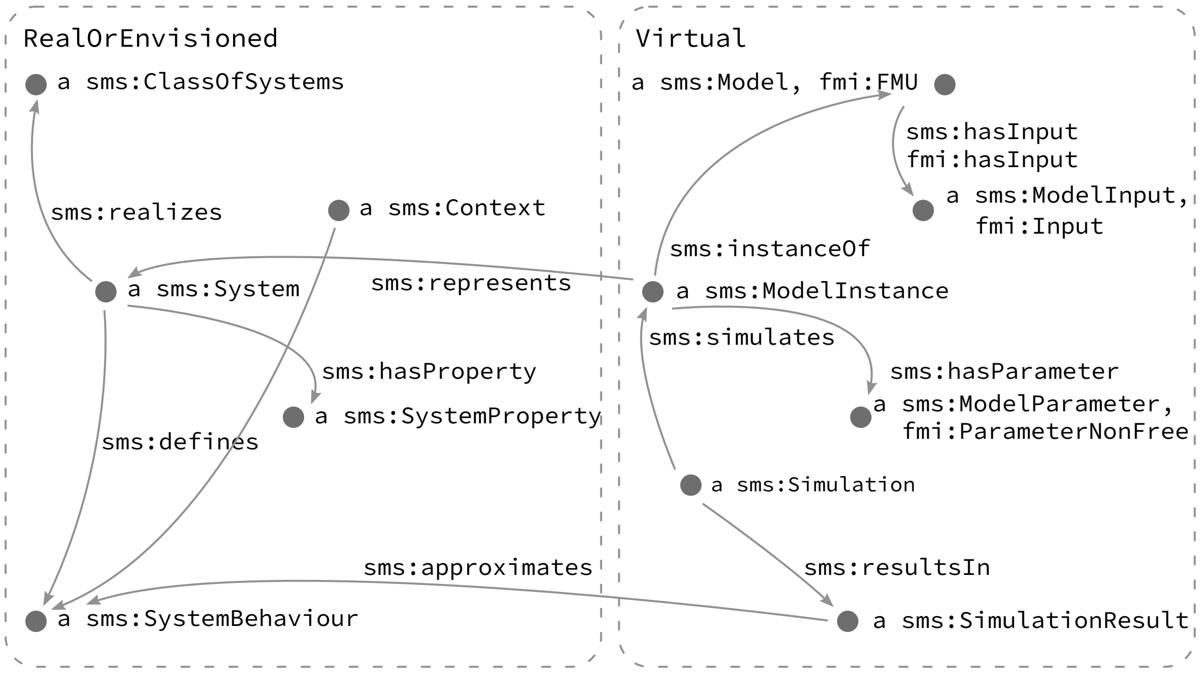

Figure 5.1: The FMI- and SMS-ontologies allow relating abstract entities of the M&S-domain to their counterparts in the real (or envisioned) world, as shown by this graph visualization.

SMS-Ontology

The core purpose of the SMS-ontology is to link the real (or envisioned) world in terms of systems, context, initial state et cetera to their abstract representations as model instances, input data, initial conditions et cetera, respectively.
Additionally, knowledge about possible relations between entities is captured, mainly in the form of the concept hierarchy and disjointness- and domain/range-statements.
For example, `ModelParameterNonFree` is a `UserInput` and disjoint with `ModelParameterFree`, which is a `ModelParameter` belonging to the `Virtual` realm, as opposed to the `RealOrEnvisioned` realm.
The RDF descriptions enabled through the use of the SMS-ontology are independent of any specific modelling formalism such as FMI.

SMS-FMI-Ontology

The purpose of this ontology is to enable deriving statements using the SMS-ontology from triples about FMUs encoded using the FMI-ontology. There are several reasons for specifying the relationships between concepts of the FMI-ontology and the SMS-ontology in a third ontology instead of as part of the SMS-ontology. Most importantly, it should be possible to develop and use the individual ontologies without unnecessary complexity/clutter, especially since a more widespread uptake of the FMI-ontology is expected than it is for the SMS-ontology—users might want to choose different ontologies for the description of systems, models and simulation whereas there is no reason to have several ontologies for describing FMUs in RDF (other than quality issues). Also, having separate ontologies helps to keep the complexity both with respect to the mental load on developers and the computational effort for reasoners minimal.

▲

The implementation of the ontologies is based on the best practices for designing FAIR ontologies suggested by Garijo, Corcho and Poveda-Villalón as part of their `FOOPS!` ontology pitfall scanner [36].

5.1.2 The `fmi2rdf`-Parser

▼ [94, section 4.1]

Given a FMU, its description in RDF should be created automatically for all triples that can be inferred from either the FMU itself or through reasoning. For this purpose, the `fmi2rdf`-package was implemented. `fmi2rdf` creates an RDF representation of the FMU based on the information in the model description, which includes metadata, variables, and parameters, including their associated types and units. Moreover, the representation created includes SHACL shapes graphs that specify the requirements for instantiating and simulating a model in terms of the requirements for an RDF graph that contains the required parameter- and input values.

In more detail, `fmi2rdf` works as follows: the `modelDescription.xml` file contained in each FMU specifies its content, structure and some metadata in a structured format. Moreover, structure and semantics of this file are defined through the FMI standard [65] and corresponding XML Schema Definition (XSD) files. Consequently, this file is used as the input of `fmi2rdf`. The function to read the descriptive XML file into a Python object provided by `FMPy`[1] is used as a starting point. From this object, the RDF representation of the FMU is built.

First, each variable, unit and type are given a URI relative to the URI of the FMU itself. Concept assertions using the appropriate concept defined in the FMI-ontology are made and, if applicable, available variable names and descriptions are captured using the predicates `rdfs:label` and `dct:description`, respectively. The metadata fields defined in the FMI standard are parsed to RDF literals with the appropriate data type using the roles defined in the FMI-ontology.

Second, assertions are made for each variable: it is linked to its declared type and unit, and additional information such as minimal/maximal/nominal or start values are captured. If a variable is declared as input or output of the FMU, this is asserted through `rdf:type`-statements as well as by pointing to them from the FMU explicitly using the `fmi:hasInput` and `fmi:hasOutput` roles.

Third, the parameters of the model that are intended to be exposed to the user are identified using one of three strategies:

- Only those parameters are selected that start with a certain string, such as the name of a group of parameters intended to be set by a user.
- Only parameters are selected that do not contain a dot in their name, which selects only top-level parameters iff a hierarchical naming scheme using dot-notation is used inside the FMU.

[1] `https://github.com/CATIA-Systems/FMPy`

Code	Persistent URL	Repository
`fmi2rdf`	—	https://github.com/UdSAES/fmi2rdf
FMI-ontology	https://purl.org/fmi-ontology	https://github.com/UdSAES/fmi2rdf
SMS-ontology	https://purl.org/sms-ontology	https://github.com/UdSAES/sms-ontology
SMS-FMI-ontology	https://purl.org/sms-ontology/fmi	https://github.com/UdSAES/sms-ontology

Table 5.1: Persistent URLs and repositories for the developed ontologies and the `fmi2rdf`-parser

– Alternatively, all parameters are selected.

Fourth, shapes that specify requirements on parameters, inputs and simulation settings are generated from relevant information contained in the model description using the SHACL ontology.

Last, a reasoner might be used to infer additional triples using ontologies that capture interrelations of concepts and roles specified in the FMI-ontology with other ontologies, such as the SMS-FMI-ontology. However, this is not the responsibility of `fmi2rdf`, but an additional step.

The `fmi2rdf`-parser is implemented in Python, using `FMPy` for reading FMU properties and `rdflib`[1] for representing and serializing the graph built. It can be used through a command-line interface (CLI) as well as from Python code and is released under the MIT licence on GitHub. Similarly, the ontologies are also released under the MIT licence on GitHub; find the links in table 5.1.

Note that the ontologies were given persistent URLs via the PURL service[2], which establishes a redirect currently pointing to the serialization of the ontology on the main branch in the GitHub-repository.

▲

5.2 The M&S hypermedia API: Concept

From a conceptual point of view, creating a cloud-native hypermedia API requires finding answers to the following questions:

- Which conceptual resources should be exposed?
- Which data model should be used, and which data should be included in the resource representations?
- How to communicate the service interface and intended real-world effect to possible users?
- How to respond to incorrect requests sent by users and what to do in case internal errors propagate to the service interface?

The last question is of high importance for a production environment. For a proof of concept software in a testing environment, one might get away with ignoring the need for proper handling of incorrect user input and service failures. Therefore, only the first three questions are discussed in detail below.

However, first a decision on the *functionality* to be exposed must be made, both on an abstract level and with respect to possible technical restrictions such as the choice for a specific model type to be supported.

▼ [94, section 3.2]

[1] https://rdflib.readthedocs.io/en/stable/
[2] https://purl.archive.org/help

With respect to the core functionality, it was decided to allow the registration of *complete and valid system models* with the service; their instantiation by setting the model parameters; their simulation subject to initial conditions and inputs; and the retrieval of simulation results.

From a technical point of view, it was decided to support *causal multiple-input/multiple-output (MIMO) blocks* for which the parameters can be set. Specifically, FMUs for co-simulation according to version 2.x of the FMI standard are supported for registration with the developed hypermedia API. The exposed resources (models, model instances, simulations and simulation results) are immutable in order to facilitate their integration in higher-level applications. However, model instances, simulations and simulation results are not stored within a service instance indefinitely (in contrast to models), but instead subject to time to live (TTL) and least recently used (LRU) caches to avoid indefinite growth of the storage allocated by an instance. Incomplete models or acausal models as well as causal MIMO blocks in non-FMU form are not supported because FMI represents the de facto standard for model exchange in the context of dynamic system simulation. This ensures widespread compatibility and allows reusing tooling created for handling FMUs, which facilitates the implementation of the software necessary to answer the research questions. Version 2.x of the FMI standard is used because FMI 3.0 had not been released at the time that the software was implemented.

▲

5.2.1 Interface Design

The entities and capabilities of the application domain M&S that are to be exposed through the service interface are complete system models with or without set parameters; the definition and execution of simulation runs; and the retrieval of simulation results. Additionally, an interface to the information held by a service instance is required.

Main Functionality

In a RESTful system, these entities and capabilities need to be expressed in terms of uniquely identifiable conceptual resources that constitute the service interface [118, p. 240]. The chosen conceptual resources to be exposed by the M&S hypermedia API are *models, model instances, simulations*, and *simulation results*.

▼ [93, section 2]

These definitions should be seen as specific to the developed software.

Models
: are well-posed system models that could be simulated once all parameters are set; but the parameters are *not* set yet.

Model instances
: are system models that *do* have all parameters set (either explicitly or by relying on default values) and could be simulated as soon as initial conditions and input values are provided. The parameters of a model instance cannot be changed; a change in parameters always leads to a new model instance.

Simulations
: combine a model instance with initial conditions, input trajectories,

a solver and the corresponding solver settings. They also have a state, for example `new`, `running` or `finished`, and link to their result iff it exists. Like model instances, simulations cannot be changed once they are created.

Simulation results
contain the actual results of exactly one specific simulation. They cannot be updated either.

Having decided which resources to expose, it must be decided which HTTP methods to allow on them. In table 5.2, the main part of the service interface in terms of meaningful combinations of resources and methods is shown. Because HTTP only offers a limited set of methods (to support the desired self-descriptiveness of messages in RESTful systems, [118, p. 242]) and because assigning URLs to actions contradicts the idea of using resources, the processes of instantiating a model instance or simulating it are triggered in the background as a result of creating a new model instance or defining a new simulation.

Both instantiation and simulation require inputs; specifically, the parameters of a model instance or the input trajectories, simulation settings and initial conditions that define a simulation. They are expected to be supplied by the service consumer in the body of the respective `POST` requests. Figure 5.2 visualizes the life cycle of the API's resources as well as the indented flow of interaction between service and service consumer. The numbers marking the transitions correspond to the last column of table 5.2.

Query Interface: Quad Pattern Fragments

Since the FAIR principle F4 asks that "(meta)data are registered or indexed in a searchable resource", it was decided to make the service itself a searchable resource for the data that it holds. Without a dedicated interface for this, there is no possibility to query the information held by an instance of the M&S hypermedia API other than retrieving all available resource representations, combining the responses into a graph and querying this graph locally. This is both inconvenient and inefficient.

[94, section 3.1]

Verborgh et al. [120] developed *Triple Pattern Fragments (TPFs)* as one specific interface that supports online querying, but keeps the cost of providing the interface low. They base their work on the observation that KGs are either not published in a queryable form (data dumps only) or subject to issues frequently observed on SPARQL endpoints, such as low discoverability, inconsistent support for all SPARQL features, high variability in query execution performance and low availability [15].

All triples matching the pattern `?subject ?predicate ?object`, where all, none, or some of the terms can be specified, are exposed by a TPF interface. The representations transferred as the result of a TPF request contain a subset of the matching triples as data (pagination is used to limit the size of the response); an approximation of the total number of matching triples as metadata; as well as a hypermedia control explaining clients how to retrieve other triple patterns of the same data set.

Method	Resource	Description	Id
`OPTIONS`	`*`	Retrieve RESTdesc descriptions in N3 serialization	0
`POST`	`/models`	Add a new model to the API-instance	1
`GET`	`/models/{model-id}`	Retrieve a model representation from the API	2
`DELETE`	`/models/{model-id}`	Delete a model representation from the API	3
`POST`	`/models/{model-id}/instances`	Instantiate a model for a specific system	4
`GET`	`/models/{model-id}/instances /{instance-id}`	Get a representation of a specific model instance	5
`POST`	`/models/{model-id}/instances /{instance-id}/simulations`	Trigger the simulation of a model instance by defining a simulation	6
`GET`	`/models/{model-id}/instances /{instance-id}/simulations /{simulation-id}`	Retrieve a representation of a specific simulation definition and its status	7
`GET`	`/models/{model-id}/instances /{instance-id}/simulations /{simulation-id}/result`	Retrieve a representation of the results of a specific simulation run	8
`GET`	`/knowledge-graph?subject=… &predicate=…&object=…&graph=…`	Query API-instance via Quad Pattern Fragment-interface	—

Table 5.2: Overview of the service interface in terms of HTTP methods, exposed resources and their meaningful combinations (incomplete)

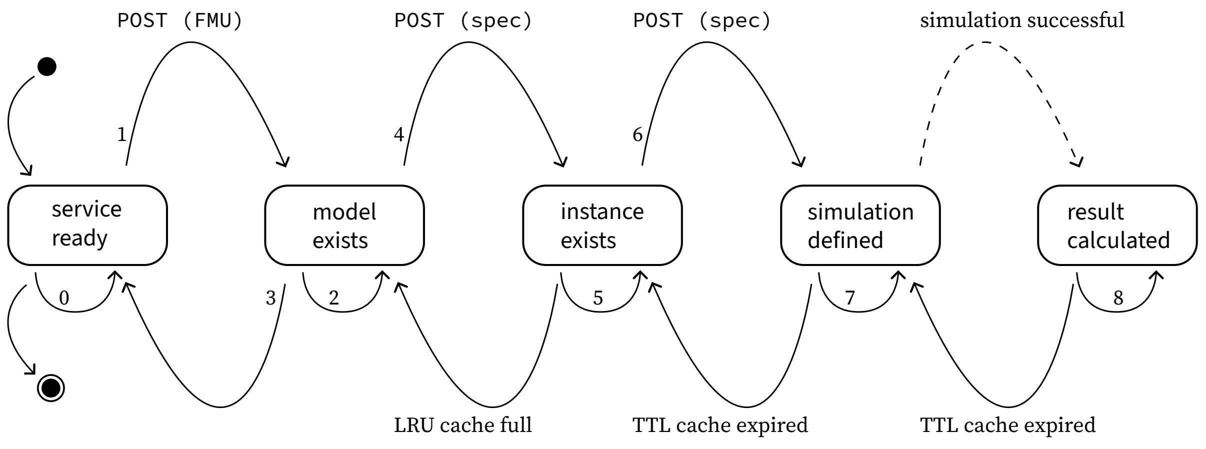

Figure 5.2: New resources are created by sending their specifications to the service instance; except for the simulation result which is added as soon as it becomes available. Its calculation is triggered when a new simulation is specified. Model instances, simulations and simulation results are not stored indefinitely to keep storage requirements limited.

Clients can still use SPARQL to formulate their queries; however, a query engine needs to decompose the SPARQL query into requests to the TPF endpoint and combine the results of these individual queries to obtain the final result [120, p. 192 ff.]. This means that the load for computing the results of a query is distributed between more intelligent clients and less powerful services compared to using a SPARQL endpoint directly.

Several advantages of the TPF interface have been observed [120, p. 203]: a reduced load on the server; better support for more clients sending requests simultaneously; and increased potential for benefiting from HTTP caches. The time to resolve a query increased, but typically stayed below 1 s until the first results were retrieved, which the authors used as the threshold for validating their hypothesis on "sufficiently fast" query execution [120, p. 186]. Moreover, TPFs are compliant with REST and thus well suited for integration into a hypermedia API. Consequently, it was decided that the service exposes a TPF interface to support querying instead of a SPARQL interface. TPFs have since been extended to Quad Pattern Fragments (QPFs) [99]. Since QPFs are supported by the software package used, they are used instead of TPFs.

5.2.2 Resource Modelling

Given the conceptual resources of the service interface and the methods that are allowed to be applied to them, the question "Which data to include in the resource representations?" arises. It is an important question because data that is not there cannot be used, neither for acting upon it nor for improving the FAIRness of the exposed resource, but it is also ambiguous. Moreover, from this question also directly follows the question "Which formats for representing the resources can be requested by clients?".

The straightforward approach to answering this question would be to use an application-specific key/value data model with a simple text-based serialization format, such as comma-separated values (CSV) or JSON; and to tailor the content of the resource representation to a specific application. The advantages of this approach are that it facilitates implementation; that it is the approach that web developers are used to; and that these simple serialization formats are natively supported by almost every software package and programming language. It is the approach taken for the REST-based HTTP-API variant of the developed API, but the rest of this section focuses on the design of the *hypermedia* representations required to implement all REST constraints.

There are two drawbacks of using an application-specific key/value data model and a simple serialization format. First, it is a defining characteristic of both SOAs and FAIRness that the service consumers are *unknown* at design-time. Therefore, the resource representations should not be tailored to one specific application. Second, simple serialization formats such as JSON are unable to meet the requirements posed on resource representations that are intended to facilitate FAIR resources and the API's use by generic software agents.

Specifically, the requirements on data model and serialization formats are that *labelled links* are supported (clients decide which links to follow based on link label); that *formal language for knowledge representation* can be used (required by FAIR principle I1); and that it is possible to *separate the actual data from auxiliary data* (facilitates implementation of clients).

A graph as the data model using RDF in conjunction with OWL ontologies, in other words using the Semantic Web technology stack, can meet all these requirements iff a serialization that supports quads, such as TriG or JSON-LD, is used.

▼ [94, section 4.2.2]

But what about the content of the resource representations? Considering three topics of data (the resource itself, data *about* the resource, pointers to related resources) and two sources of data (the humans that create a digital asset such as a dynamic system model or the software used to process it), two strategies for adding relevant triples to resource representations emerge.

First, triples that can be generated automatically, should be generated automatically. This likely concerns triples encoding the data itself or triples encoding metadata that is already available or gets created as part of a digital process. The ontologies used are likely of methodological (for example the FMI-ontology) or descriptive nature (DCT, The PROV Ontology (PROV-O)[1], ...); the level of detail is chosen by the programmers. For example, data about the origin of a simulation result should be recorded throughout the different parts of the developed software solution as suggested in [60]. Specifically, it might be interesting to know that a certain version of `FMPy` was used for simulation on behalf of a certain version of the worker component, which could be encoded using PROV-O.

Second, triples that cannot readily be generated from source files such as additional metadata; knowledge that only exists in the minds of humans; and pointers to related relevant knowledge should be added by the human that created, in this case, the model to be exposed through the developed hypermedia API. The level of detail depends on the envisioned applications; questions that could guide users include "what does the model represent?"; "which question(s) was it intended to solve?"; "where does the knowledge encoded (equations, parameters, defaults) come from?"; and "what other resources might future users of the model want to look at?". To encode such information, ontologies to link the virtual world to the real or envisioned world such as the SMS-ontology and referential ontologies such as the Sensors, Observations, Samples, Actuators (SOSA)-ontology[2] in conjunction with DBpedia[3] would likely be useful. Technically, these triples could be stored inside the FMU as vendor annotations [65, sec. 2.2.6] and then added to the representation of the FMU in RDF by the `fmi2rdf`-package.

▲

The content of a resource representation can be categorized in data, controls, context and metadata [113]. The differences between these categories are explained using an (abbreviated) exemplary representation of a model resource, serialized using the TriG syntax [13], shown in listing 5.1. The

▼ [94, section 4.2.2]

[1] `http://www.w3.org/TR/prov-o/`
[2] `https://www.w3.org/TR/vocab-ssn/`
[3] `https://www.dbpedia.org/`

```
@prefix api: <http://example.com/vocabulary#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix fmi: <https://purl.org/fmi-ontology#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix hydra: <http://www.w3.org/ns/hydra/core#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix sms: <https://purl.org/sms-ontology#> .
@prefix spdx: <http://spdx.org/rdf/terms#> .
@prefix var: <http://example.com/models/6157f34f/variables#> .
@base <http://example.com/models/6157f34f> .

# Data and metadata (about the resource itself, in the default graph)
<> a fmi:FMU, sms:Model ;
    fmi:hasInput var:temperature, var:windSpeed,  ... ;
    fmi:hasOutput var:powerDC, var:totalEnergyDC, ... ;
    fmi:hasParameter var:panelArea, var:panelTilt, ... ;
    fmi:modelName "PhotoVoltaicPowerPlantFMU" ;
    fmi:fmiVersion "2.0"^^xsd:normalizedString ;
    prov:wasAttributedTo <https://orcid.org/0000-0002-4006-8582> ;
    spdx:declaredLicense <http://spdx.org/licenses/MIT> ; ... .

# Context and controls (in dedicated named graph(s))
<#about> {
    # Context: Metadata about the resource representation
    <#about> foaf:primaryTopic <> .
    <> dct:created "2021-11-29T12:31:01Z" .

    # Controls: What can I do with this resource?
    <> hydra:supportedOperation [
      a hydra:Operation ; hydra:method "DELETE" ; ...
    ] , ... .

    # Controls: Links _within_ API-instance; to enable HATEOAS
    <> api:home </> ;
       hydra:collection </models> ; api:allModels </models> ;
       hydra:collection </models/6157f34f/instances> ;
       api:allInstances </models/6157f34f/instances> ;
       sms:instantiationShape <#shapes-instantiation> ; ... .

    # Context: Suggestions for related resources outside API-instance
    <> prov:influenced <http://doi.org/10.3389/fenrg.2021.639346> ; ... .
}

<#shapes> {
    <#shapes-instantiation> a sh:NodeShape ; sh:targetNode [ ] ;
                            sh:property [...] ; ... .
}
```

Listing 5.1: TriG-serialization of a model representation (abbreviated)

triples that encode that the resource identified by the URI of this document is a `fmi:FMU` and a `sms:Model` and link it to its inputs, outputs and parameters (line 14 to 17), represent the data part of the resource representation. The triples in line 18 to 21 can be seen as both metadata because they describe the FMU and as data because they are part of the conceptual resource that is exposed.

In the example, the separate graph used to distinguish data from metadata, context and controls, is named `<#about>`. In it, first some metadata about the resource representation is provided, such as when the resource was created (context). Next, possibilities for interacting with *this specific resource* are communicated to the client using elements of the Hydra core vocabulary [56] in lines 30 to 32. Then, links to related resources provided by the *same* the API instance are offered to the client in lines 35 to 39 (controls). Hydra is used as one example of how these controls might be encoded. Last, non-committal suggestions for resources outside the API's context that might also be of interest are made in line 42; these implement the FAIR principle I3, "(Meta)data include qualified references to other (meta)data", and provide more context.

In general, more detailed resource representations would increase the possibilities for finding and interacting with resources. Currently, more elaborate strategies to provide "rich" metadata, such as the automatic recording of a provenance trail, are not implemented. However, even resource representations that are not very detailed suffice to validate the hypotheses of this work as shown in chapter 6 and section 7.1. Note that due to the ambiguity of the question of which triples to provide in a given resource representation, it was aimed to provide at least one meaningful statement for each category at this point in time. ▲

5.2.3 Advertising Service Capabilities

▼ [94, section 3.2] When navigating websites, humans rely on expectations based on experience and intuition to decide which controls offered by the website will most likely lead them to their goal [112, p. 39]. In other words, humans establish a plan based on implicit information, hoping and assuming that it will successfully resolve. Software agents require a plan based on explicit information to determine if they can meet their goal [112, p. 40]. ▲ One goal for the M&S hypermedia API is increased machine-actionability (H2.3). Consequently, relying on the intuition of humans is not an option. In this section, two approaches for advertising the capabilities and describing the interface of a service are shown.

The first approach is the use of a commonly agreed upon format for describing a service interface, such as the OpenAPI Specification (OAS)[1]. This represents the state of technology for APIs in the Web, and it is used by the REST-based HTTP-API variant of the service. However, it promotes tight coupling and is not compliant with the REST constraints.

▼ [94, section 1.2] The programming of clients against a static service interface description at design-time is especially problematic when exposing M&S capabilit-

[1] https://www.openapis.org/what-is-openapi

ies through an API: for every model, the parameters for instantiation and simulation are different. Static service interface descriptions consequently either have to be kept so generic that they cannot realize their usefulness, or be re-generated every time a model is added to an instance of the API [93, p. 395]. This would entail that programmers had to first add a model to the API instance they plan to use before they could program the subsequent requests, which is inefficient and would make the use of the API for a large number of different models prohibitively expensive.

The second approach is to *not* have a separate documentation of the service interface, but to integrate metadata, context and controls into the resource representations, as discussed in the previous section. This enables the HATEOAS principle and loose coupling.

The service interface consists of controls which allow consumers to interact with the service; however, consumers need to know which effect using a control will have in order to decide whether to use it. Next, means to communicate the available controls are discussed first before elaborating on how to communicate their effects to potential service consumers. Last, encoding requirements on data to be sent as part of requests is discussed. All discussions focus on the second approach and only briefly mention the use of the OAS because the latter is not suitable to attain the goals of this work and only serves as a baseline for comparison.

Communicating Controls

In the OAS document, each applicable HTTP-method is documented in terms of request parameters and expected responses for every resource (listing 5.2).

For the hypermedia API variant of the service, there are different ways of communicating the available controls within the resource representations. All of them must enable the service consumer to create a fully specified HTTP request, which consists of the HTTP method, the request URL, headers, authorization information and possibly parameters in the body and/or path of the request. Additionally, a hint at the consequences of using a control should be given.

The simplest case of a control is a link that is intended to be dereferenced, in other words a fully specified URL which is intended to be used in a `GET` request. For example, the triple in line 42 of listing 5.1 points to a research paper that was influenced by the model. The nature of the resource to be resolved is indicated by the *link relation*. For humans, the link relation corresponds to the label of the link (for example: in a list of articles, an article's title links to the full version); for software agents, the predicate that relates a subject node to an object node encodes the link relation. Commonly used link relations are specified in the Hydra Core Vocabulary [56]. For example, the predicate `hydra:collection` points to "collections somehow related to this resource"[1]. However, more specific predicates might be required to guide a client through the use of a hypermedia API. For example, it might be necessary to distinguish between a

[1] https://www.hydra-cg.com/spec/latest/core/#hydra:collection

```
1   {
2     "paths": {
3       "/models/{model-id}": {
4         "get": {
5           "summary": "Retrieve a model representation from the API",
6           "responses": {
7             "200": { "$ref": "#/components/responses/ModelRepresentation" },
8             "400": { "$ref": "#/components/responses/SchemaValidationFailed" }
9           }
10        },
11        "parameters": [ { "$ref": "#/components/parameters/ModelId" } ]
12      }
13    },
14    "components": {
15      "parameters": {
16        "ModelId": {
17          "name": "model-id",
18          "in": "path",
19          "required": true,
20          "example": "29dc05e6-1e05-4076-bbe5-79d4e0d2770c",
21          "schema": { "$ref": "#/components/schemas/ModelId" }
22        }
23      },
24      "responses": {
25        "ModelRepresentation": {
26          "description": "A representation of a model",
27          "content": {
28            "application/json": {
29              "schema": { "$ref": "#/components/schemas/Model" }
30            },
31            "application/octet-stream": {
32              "schema": { "type": "string", "format": "binary" }
33            }
34          }
35        }
36      },
37      "schemas": {
38        "ModelId": {
39          "description": "The UUIDv4 of the model",
40          "type": "string",
41          "pattern": "^[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-4[0-9A-Za-z]{3}-[89AB..."
42        },
43        "Model": {
44          "type": "object",
45          "properties": {
46            "modelName": { "type": "string" },
47            "description": { "type": "string" },
48            "fmiVersion": { "type": "string" },
49            "generationTool": { "type": "string" },
50            "generationDateAndTime": { "type": "string" }
51          }
52        }
53      }
54    }
55  }
```

Listing 5.2: Excerpt of the OAS (serialized as JSON). In natural language, it can be summarized as follows: for retrieving a representation of a model from the API, a `GET` request needs to be sent to `/models/{model-id}`. The model identifier is a string that matches the pattern for a UUIDv4. The response to a successful request is either a binary representation of the model or a JSON-representation, depending on the hypermedia type requested. The JSON-representation consists of five string properties (lines 46 to 50). If the consumer sends a bad request, a response with the code `400` will be sent (details not shown in the excerpt).

collection of models and a collection of model instances. To do so, it was decided to use a vocabulary specific to the developed API, exposed as part of the API at `/vocabulary#`, as shown in lines 36 to 38 in listing 5.1.

In addition to specifying common link relations, the Hydra Core Vocabulary also allows encoding controls, as briefly demonstrated in lines 30 to 32 in listing 5.1. A third alternative for encoding controls is provided by the HTTP-ontology[1], which is used within the RESTdesc descriptions described next.

Communicating the Effect of Controls (RESTdesc)

Deliberately interacting with controls offered by a service is only possible when their effects are known before and without executing them.

For the REST-based HTTP-API variant, these consequences of using a control are explained to humans in natural language as part of the OAS (listing 5.2, line 5). The level of detail depends on the programmer writing the OAS, and may or may not include information about the role a resource can play in interacting with the service.

For communicating the effect of controls to software agents, link relations alone do not allow to predict the effect of state-changing operations such as `POST` requests. In other words, controls answer the question of how to leave one state, but not what the next state will be. Consequently, a description is needed that communicates which transitions are possible in a given application state and what the effects of these transitions in terms of changes to the shared state are. The PPA relies on RESTdesc descriptions to communicate this information; therefore, the choice for RESTdesc is a direct result of the choice for using the PPA and alternatives, such as ontologies for service description, were not regarded.

▼ [94, section 3.2]
▲

RESTdesc descriptions are N3 [81] rules that communicate the existence and form of an HTTP request that allows transitioning from one resource state, the precondition, to another resource state, the postcondition. Syntactically, the format

```
{ <precondition> } => { <HTTP-request> <postcondition> }.
```

is used to encode this information.

▼ [94, subsection 4.2.3]

By means of a formal proof, it can be shown that a goal can be achieved *without* executing any of the requests (instead, the proof assumes that they succeed). In addition to determining the achievability of a goal, the proof also shows which requests out of all possible requests by all available hypermedia APIs contribute to achieving the goal, thus representing a high-level plan.

As an example, see the RESTdesc description for retrieving a TriG-serialization of a model representation shown in listing 5.3. Line 11 states the precondition, followed by the description of the HTTP request in lines 15 to 21 that, iff successful, will result in the postcondition (lines 23 to 27). The RESTdesc description contains universally quantified variables,

[1] http://www.w3.org/2011/http

prefixed by a question mark, and existentially quantified variables using `_:` as prefix. Universally quantified variables in the request description and the postcondition must also occur in the precondition and thus will be known when the rule is about to be applied. In contrast, existentially quantified variables in the postcondition convey an expectation of what the API's response will contain. For a detailed formal definition of RESTdesc, please refer to Verborgh et al. [119, section 4.3].

In natural language, the first rule shown in listing 5.3 reads as follows: "Given a URI denoting a `sms:Model` (line 11), a representation of the model can be obtained by sending a `GET` request to the model's URI with the `Accept` header set to the media type `application/trig` (lines 15 to 21). The model representation returned as a response will link the model to a node via an `api:allInstances` predicate and to another node through a `sms:instantiationShape` predicate (lines 23 to 24). At this point, we do not know anything about the nature of the first node, but we know that the second node will be a `sh:NodeShape` with a third node (unspecified for now) as target node (line 27)". The predicates `api:allInstances` and `sms:instantiationShape` and the nodes they point to gain meaning through their use in other RESTdesc descriptions as this use explains their role in transitioning between different application states.

Note that most rules cannot be directly instantiated because values for variables in the RESTdesc description only become known at run-time. This is a desired characteristic: the rules are only for high-level planning and determining whether a goal can be achieved in principle. The postcondition is an incomplete view on the resource representation to be expected, focused only on triples relevant to guiding clients through the application ("what terms are needed for triggering state transitions?"). The specific interaction between a client and the service is then driven by client selection of service-provided options ("what are the values of the needed terms?") at run-time as intended in RESTful systems according to the HATEOAS constraint.

For new hypermedia APIs, the RESTdesc rules are written by the programmers. Once they exist, the question becomes how to communicate their content to potential clients with minimal overhead. HTTP provides the `OPTIONS` method, which "allows a client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action" [32, section 9.3.7]. It is specific to the URL to which the `OPTIONS` request was sent, but the specification also states that an "`OPTIONS` request with an asterisk ("*") as the request-target [...] applies to the server in general rather than to a specific resource". Consequently, the M&S services created in the context of this thesis serve the RESTdesc rules for all relevant interactions in one N3 document transferred as the response to an `OPTIONS` request at the path `*`.

Requirements on Request Bodies

If there is a request body to be sent, there are typically restrictions posed on its content which should be checked upon retrieval by an API, and the request should get rejected if the restrictions are not fulfilled. The require-

```
@prefix http: <http://www.w3.org/2011/http#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix fmi: <https://purl.org/fmi-ontology#> .
@prefix sms: <https://purl.org/sms-ontology#> .
@prefix api: </vocabulary#> .


# Get a model representation
{
    ?model a sms:Model .
}
=>
{
    _:request http:methodName "GET" ;
              http:requestURI ?model ;
              http:headers [
                  http:fieldName "Accept" ;
                  http:fieldValue "application/trig"
              ] ;
              http:resp [ http:body ?model ] .

    ?model api:allInstances _:allInstances .
    ?model sms:instantiationShape _:instantiationShape .

    _:instantiationShape a sh:NodeShape ;
                         sh:targetNode _:parameterSet .
} .


# Instantiate a model
{
    ?model a sms:Model ;
           api:allInstances ?allInstances ;
           sms:instantiationShape ?instantiationShape .

    ?instantiationShape sh:targetNode ?parameterSet .
}
=>
{
    _:request http:methodName "POST" ;
              http:requestURI ?allInstances ;
              http:body ?parameterSet ;
              http:headers [
                  http:fieldName "Accept" ;
                  http:fieldValue "application/trig"
              ] ;
              http:resp [ http:body ?modelInstance ] ;
              http:resp [ http:headers [
                  http:fieldName "Location" ;
                  http:fieldValue ?modelInstance
              ]] .

    ?modelInstance a sms:ModelInstance ;
                   sms:instanceOf ?model .
} .
```

Listing 5.3: RESTdesc descriptions for retrieving a model representation and instantiating a model

```
<#shapes-instantiation> a sh:NodeShape ;
  sh:targetNode :parameterSet ;
  sh:property [
    sh:path :latitude ;
    sh:node [
      a sh:NodeShape ;
      sh:property [
        sh:path sms:isValueFor ;
        sh:hasValue var:latitude ;
      ] ;
      sh:property [
        sh:path qudt:numericValue ;
        sh:maxInclusive 90 ;
        sh:minInclusive -90 ;
      ] ;
    ] ;
    sh:minCount 1 ; sh:maxCount 1 ;
  ]
.
```

shapes graph

```
:parameterSet a sms:ParameterSet ;
  :latitude [
    sms:isValueFor var:latitude ;
    qudt:numericValue 49.31659 ;
    qudt:unit unit:DEG
  ] ;
  :longitude [..] ;
  :panelArea [..] ;
  :
  :
.
```

data graph

Figure 5.3: The excerpt of a SHACL shapes graph on the left communicates that its target node must point to exactly one object node via the `:latitude` -predicate. This object node must point to `var:latitude` via `sms:isValueFor` and to a numeric value in the range −90 to 90 via the `qudt:numericValue` -predicate. On the right, the corresponding data graph is shown.

ments on the request body can be encoded on a syntactical level or on a semantic level.

For HTTP-APIs documented according to the OAS, JSON Schema[1] is used to encode requirements on request bodies on the syntactical level. There is no example of this in listing 5.2 because the shown request doesn't require a body to be sent; however, the schemata in lines 37 to 53 provide an example of the syntax used.

For the Semantic Web, the W3C recommends the use of the Shapes Constraint Language (SHACL) [51] for imposing constraints on graphs. A *shapes graph* encodes the requirements on the *data graph*; the `sh:targetNode` -predicate is used to point from the shapes graph to the data graph (figure 5.3).

The existence of requirements on request bodies in the form of SHACL shapes graphs is integrated in the RESTdesc description as follows: the triples

```
?model sms:instantiationShape ?instantiationShape .
?instantiationShape sh:targetNode ?parameterSet .
```

in the precondition of a rule in conjunction with the triple

```
_:request http:body ?parameterSet .
```

▼ [94, section 4.3]

in the specification of the HTTP request to be sent (in the same rule; lines 35, 37 and 43 in listing 5.3) communicate that some content `?parameterSet` is to be sent as the request body, and that there is a `sh:targetNode` relation from some node `?instantiationShape` to the content. This relation im-

[1] https://json-schema.org/, https://spec.openapis.org/oas/latest.html#schema-object

plies that `?instantiationShape` is a SHACL shape, and that the content is specified by and must conform to the constraints contained in this shape.

Note that the actual specification of the shape is never included in the RESTdesc rules. The shape is generated by the service upon addition of a model, and it is then communicated at run-time as part of the resource representations exchanged: it depends on the resources, therefore it cannot be known at design-time when the RESTdesc rules are formulated by the programmers implementing the service. ▲

However, the RESTdesc descriptions need to communicate which resource representation contains the definition of the shapes graph, such that the PPA can obtain the shapes graph *before* preparing the data accordingly as part of the service interaction (see section 5.4.2). For this example, this is done in the RESTdesc rule for retrieving a model representation (lines 24 to 27, listing 5.3).

5.3 The M&S hypermedia API: Realization

Although the implementation of software is prone to take a considerable amount of effort and time, it is not always interesting from an academic point of view. Mostly, software stack and supporting technology already are established. Programming is necessary to show feasibility of one's ideas, but it is only a tool.

Nonetheless, the choices with regard to software architecture, implementation strategy, and operations affect the characteristics of the software and are thus important when attempting to implement a CNA. Consequently, the chosen software architecture for the MSaaS-implementation; requirements on the FMUs supported; and the chosen approach to software development and -operations (DevOps) are outlined in this section.

5.3.1 Software Architecture

▼ [94, section 4.2.1]

The implementation of the service's functionality is structured in four main parts (figure 5.4): the *API* component exposes the service interface and handles incoming requests by passing them on to the actual implementation of functionality as well as by sending representations of the resources in the desired format. The *worker* component performs all tasks that are specific to a model format, such as generating an RDF representation from a FMU that includes the SHACL shapes graphs for instantiation and simulation or the actual simulation. API and worker are connected by a task queue consisting of a *message broker* that transfers task representations from the API to available worker instances and a *result backend* that propagates serializations of the task results back to the API. ▲

▼ [93, section 3.1]

The specific software components and libraries used for implementation were chosen based on the criteria that they are well-suited for the job; free/libre and open-source software (FLOSS); represent the state of the art; and that their use avoids re-implementing functionality that already has stable implementations.

The HTTP-API possibly receives many requests at once, most of which result in requests to storage or other services which are operations

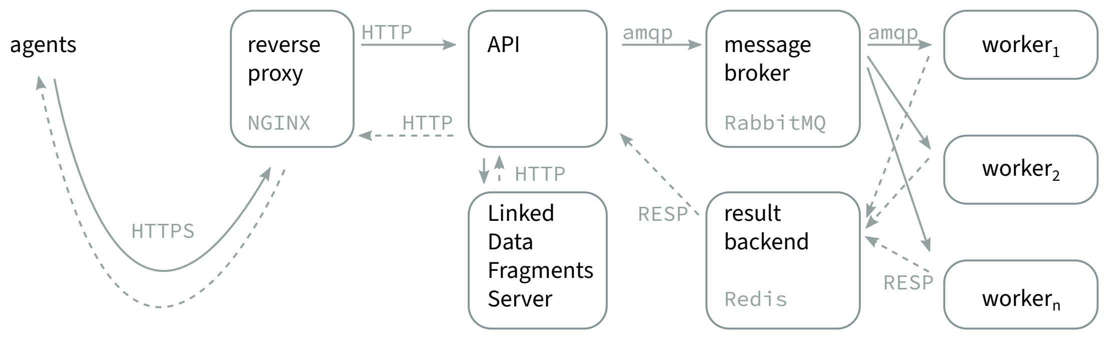

Figure 5.4: The implementation is structured in distinct API- and worker components which exchange data via queues; a reverse proxy provides a HTTPS connection to users. An instance of https://github.com/LinkedDataFragments/Server.js enables querying (proxied through the API).

that are very slow compared to pure computations, meaning that significant amounts of time are spent waiting. Therefore, Node.js[1] was chosen as the programming language as it provides excellent support for non-blocking input-output (IO) operations using `promises` and the `async/await`-syntax. Also, it is commonly used for implementing HTTP-APIs and consequently offers many useful libraries that support implementation, such as the Express-framework[2] and the `openapi-backend`[3]. Incoming requests are checked for validity against their schema in the OAS. Valid requests are then propagated to the appropriate handlers, which alter or retrieve resource state and enqueue simulation requests if necessary.

The internal representation of a simulation job couples the worker implementation to the API. Workers retrieve these representations from the task queue, which is implemented using `Celery`[4], using `RabbitMQ`[5] as the message broker. As a result of only coupling API and worker through the task representation, the workers can use Python[6] as programming language. This enables the use of the `pandas`[7] package for representation and manipulation of time series, and allows using `FMPy` for simulating FMUs.

`FMPy` was chosen because it can be used natively from within a Python environment; because it is actively maintained and developed under an open-source licence; and because `FMPy` and its dependencies can be installed easily, also as part of a container image.

Worker instances can be added or destroyed according to demand and jobs are automatically distributed across all worker instances that are available. Upon finishing a simulation job, the results are propagated back to the API component using `Redis`[8] as the result backend.

▲▼ [94, section 2.2]

There are several desirable consequences of this separation of concerns. First, the computing power can be scaled by adding more worker

[1] https://nodejs.org
[2] https://expressjs.com
[3] https://github.com/anttiviljami/openapi-backend
[4] https://github.com/celery/celery
[5] https://www.rabbitmq.com
[6] https://www.python.org
[7] https://pandas.pydata.org
[8] https://redis.io

instances and the existence of a queue enables "shaving" peaks in demand. Second, producer and consumer of tasks can be implemented in different languages to account for the different nature of their respective purpose. API and worker are available at `https://github.com/UdSAES/simaas-api` and `https://github.com/UdSAES/simaas-worker`, respectively; subject to the conditions of the MIT licence.

To support querying, a QPF interface providing read-only access to all triples relating to models, model instances and their properties is exposed at `/knowledge-graph`. It is implemented as follows: the API component adds new triples to a file acting as a triple store. This triple store is read by an instance of the `Linked Data Fragments Server`[1], to which the API proxies requests at the path `/knowledge-graph`.

▲

5.3.2 Restrictions on Supported FMUs

▼ [93, section 3.2]

In order to facilitate the implementation of the envisioned software, additional restrictions are imposed on the FMUs concerning their parameterization; the supported platforms for which binaries must exist; and the definition of inputs, outputs and parameters to be exposed via the API. Note that none of these restrictions impose limits on the actual models or their simulation; they merely represent a concretization of the format supported by the developed software.

Schmitt et al. [85] investigated different possibilities to parameterize models in Dymola with respect to their subsequent export as FMU. In their paper, they describe a method that "becomes favourable if the user wants to exchange whole data sets of one and the same model" [85, section 3.3], which is exactly the case for the SIMaaS-implementation. In short, parameters inside the model are set by inter-component references to a record. This record has a parameter `filename`, which must be set to the path of a file containing the values. All actual model parameters are set by reading this file during model initialization. This can, for example, be achieved using the `DataFiles` package distributed with Dymola or one of the functions provided in `Modelica.Utilities`.

The second requirement is that inputs and outputs of the models must be listed as such in the `modelDescription.xml` file of the FMU because the generation of the schemata for the trajectories that a service consumer needs to supply/can expect as a result, which are both part of the OAS (as JSON schemata) and part of the RDF-based resource representations (as SHACL shapes graphs), relies on this information.

Last, the FMU must contain binaries for GNU/Linux as the containers are intended to be deployed on GNU/Linux host systems.

▲

5.3.3 DevOps

The software developed as part of this thesis is not intended for production environments, but for demonstrating feasibility. Nonetheless, the DevOps mindset was adopted because it is a proven approach to dealing with the

[1] `https://github.com/LinkedDataFragments/Server.js`

complexity of developing CNAs in a SOA; an agile approach to project management was taken because it is suitable for projects with high uncertainty; and best practices were incorporated because they help with getting things right from the start.

Obviously, setting up processes and tools to manage DevOps requires work. However, it was found that this work was helpful and valuable because it fostered *understanding* challenges associated to CNAs; it inspired *confidence* in the chosen setups; and it *documented* the configurations in an executable manner, making them *reproducible*.

The specific measures taken to support DevOps and facilitate the creation of high-quality software are briefly listed below, with the intent to give a high-level overview without going into details.

- All source code is versioned using `git`[1] as the version control system (VCS). This ensures that changes to the source code can be tracked, and it can be used to ensure that only "clean" versions of a software (without any unregistered changes) are deployed.
- A feature branch workflow[2] is adopted to facilitate parallel development of different features; and to help with releasing versions. For example, all changes integrated into the `dev`-branch get automatically deployed privately, whereas only changes merged into the main branch get deployed on the public server and tagged as a versioned release.
- Semantic versioning[3] is used for all software and ontologies.
- It is attempted to provide good `README` documents[4] in all public repositories.
- Licences are declared using Software Package Data Exchange (SPDX)[5] identifiers and roughly following the suggestions of the REUSE initiative[6].
- All dependencies of a software are explicitly declared in dependency manifests, such as `package.json` for Node.js, `requirements.txt` for Python, and build instructions (`Dockerfile`).
- All source code is automatically formatted (using `black`[7] for Python and JavaScript Standard Style[8] for Node.js) to ensure a consistent reading experience and to follow best practices for a given programming language.
- All configuration is stored in environment variables, never in the source code. This enables changing the configuration for different deployments/environments without changing the code; and it ensures that secrets, such as access tokens or passwords, are not leaked by committing them in the VCS.

[1] https://git-scm.com/
[2] https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow
[3] https://semver.org/
[4] https://www.makeareadme.com/
[5] https://spdx.dev/
[6] https://reuse.software/
[7] https://github.com/psf/black
[8] https://standardjs.com/

- Relevant events that occur while running a software (start up; file access; incoming/outgoing requests; …) are written to `stdout` in a *structured* form, for example as JSON objects. In other words, log entries directly expose fields such as time, role (API/worker), severity level, HTTP request parts, hostname/container identifier, …. This means that these can be directly used for filtering when reviewing logs instead of relying on parsing or just reading textual log messages for finding relevant information.
- Logs are picked up by `fluentd`[1] as a log collector and then stored in an instance of Elasticsearch[2], which is accessed by a Kibana[3]-instance that allows human users to display, filter and visualize logs across all deployed software instances on all servers.
- Important parts of the software are tested through both unit and API tests.
- The software and all of its dependencies are packaged into container images which can then be executed as containers (in other words, the images built represent the executable form of the software; a running instance is called container). An image registry is used to store and distribute images for deployment.
- The two available servers are configured such that they provide a development/staging environment which is not accessible publicly, and a production environment which is accessible from the outside world. In the private environment, the newest versions of the services as well as *auxiliary services* such as the CI/CD server, a container image registry, an instance of the log stack et cetera, are deployed. In the production environment, more stable versions of the service are made available publicly.
- Given a GNU/Linux OS installation on the servers, all additional software installation and -configuration is managed using `ansible`[4]. Essentially, this means that the desired state of a software is defined declaratively in text files using the YAML Ain't Markup Language (YAML) syntax[5], and `ansible` then ensures that the actual state of a server matches the desired state. This documents the configuration; makes it reproducible; and facilitates that development/staging and production environment are as similar as possible (dev/prod parity). All files used by `ansible` are managed in a `git` repository. This approach is known as Infrastructure as Code (IaC).
- `ansible` is also used for deploying all services; and for defining and storing different configurations for different environments. These deployment instructions are accessed by the CI/CD server.
- In addition to the recommendations of the twelve-factor app methodology [123] already addressed (config; structured logs; dev/prod parity), also the recommendations for handling backing services; processes; concurrency; disposability; port binding; separating build,

[1] https://www.fluentd.org/
[2] https://www.elastic.co/elasticsearch/
[3] https://www.elastic.co/kibana
[4] https://www.ansible.com/overview/how-ansible-works
[5] https://yaml.org/

release, and run stages of CI/CD pipelines; and admin processes are realized.
- From a project management perspective, the work was organized in an agile manner, inspired by the Scrum framework[1] and supported by the Taiga[2] web app.

To summarize, software development and -operations was guided by the twelve-factor app best practices, building upon state-of-the-art open-source components and libraries. Containers were used as deployment units, bundling the software and its explicitly declared dependencies. Upon each commit to the main branches of the VCS, a pipeline that automatically tests, builds and deploys the software was triggered. Structured logs were collected and stored in a way that allowed searching, filtering and reviewing them efficiently. Two servers were used, serving both as separate private development environment and public production environment, and as the constituents of a clustered elastic platform. The setup and deployment of all software was managed declaratively. Almost all developed software was released under a permissive open-source licence (see section B.2).

5.4 The Pragmatic Proof Algorithm

▾ [94, section 3.2]

In the hypotheses of this work, it is suggested that the service concept described in section 4.2 leads to increased machine-actionability (H2.3) and FAIRness (H2) of the exposed M&S capabilities. The former of these hypotheses is validated by example, meaning that H2.3 is validated iff a generic software agent is able to achieve a goal by using the developed service without being specifically programmed to the service interface. The algorithm implemented by the generic software agent itself is *not* the focus of this work; it is just a means to demonstrate the validity of H2.3. The requirements on this algorithm are that it has been shown to successfully compose and execute hypermedia APIs, and that it is described in enough detail that it can be implemented. The Pragmatic Proof Algorithm by Verborgh et al. [119] fulfils these requirements and was thus chosen for this work without performing an in-depth literature research on other possible algorithms first.

▴

5.4.1 The Original Pragmatic Proof Algorithm

▾ [94, section 4.3]

Creating a proof from all RESTdesc descriptions, the goal to be achieved and the initial knowledge may show that the goal can be reached, but it is not sufficient for actually achieving the goal. An algorithm is needed that, in addition to triggering the creation of the proof, also executes fully specified HTTP requests in the correct order; parses the responses and matches the obtained knowledge with the high-level plan in form of the proof (in other words replaces the variables in the RESTdesc rules with values once they become known). This is what the Pragmatic Proof Algorithm (PPA)

[1] https://scrumguides.org/
[2] https://taiga.io/

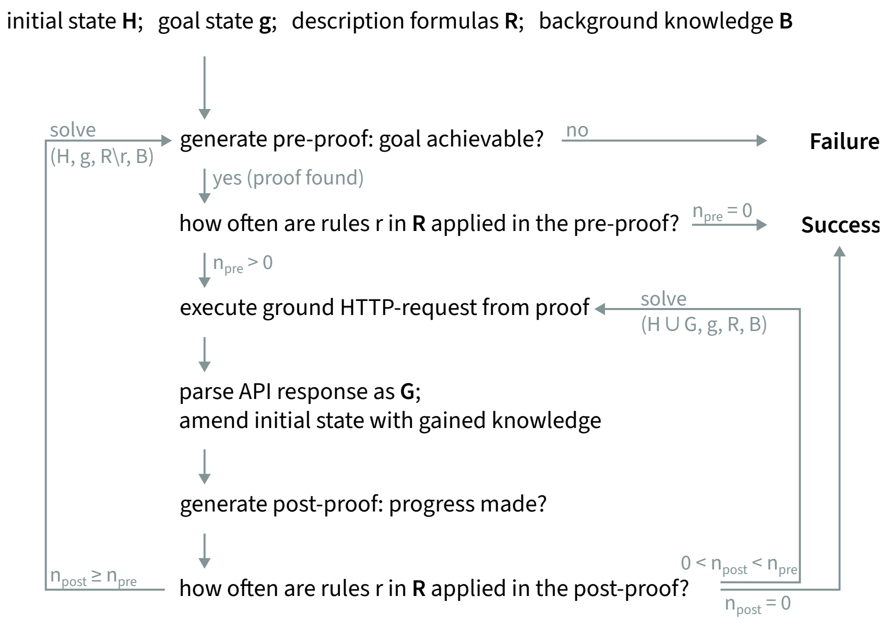
initial state H; goal state g; description formulas R; background knowledge B
solve (H, g, R\r, B)
generate pre-proof: goal achievable?
no
Failure
yes (proof found)
how often are rules r in R applied in the pre-proof?
$n_{pre} = 0$
Success
$n_{pre} > 0$
execute ground HTTP-request from proof
solve (H ∪ G, g, R, B)
parse API response as G; amend initial state with gained knowledge
generate post-proof: progress made?
$n_{post} \geq n_{pre}$
how often are rules r in R applied in the post-proof?
$0 < n_{post} < n_{pre}$
$n_{post} = 0$

Figure 5.5: The Pragmatic Proof Algorithm (PPA) [119] solves hypermedia API composition problems.

by Verborgh et al. [119, p. 34] does. Moreover, it needs to be ensured that correctly shaped input data is provided iff necessary.

The PPA is visualized in figure 5.5. It can be summarized as follows: first, the initial state H, the goal state g, the set R of all RESTdesc description formulas $r_1 \dots r_i$ and, optionally, background knowledge B are collected. Initial state and background knowledge are collections of triples; they could be general or domain-specific. The goal state is an *N3 filter rule*. Filter rules are instructions for an N3 reasoner to find all triples matching the pattern in the first part of the filter rule and to represent them according to the pattern in the second part of the rule, meaning that both parts of the rule can be identical [112, p. 51]. Together, H, g, R, and B form an *API composition problem* [119, p. 22] (see section 6.2.2 for examples). After collecting the constituents of the composition problem, the initial pre-proof is generated. If a proof is found, the goal is seen as achievable, and it is counted how many times RESTdesc rules are applied in the proof, which corresponds to the number of requests necessary to achieve the goal if all requests succeed. To begin moving towards the goal, the first fully specified request found in the proof is executed, and the response is parsed as a graph G. Since requests can fail or return content irrelevant to achieving the goal despite their RESTdesc description, a post-proof is generated to check the progress towards meeting the goal. If the number of rule applications in the post-proof is lower than in the pre-proof, progress was made and the post-proof is used as the pre-proof in the next iteration. If not, the rule that describes the request made is eliminated and the response is disregarded in the next iteration. The PPA terminates if either no proof can be found (failure) or no rule is applied in the pre-proof, which is the case if the proof shows that the goal was already met (success).

Note that no part of the PPA is specific to any hypermedia API: given the RESTdesc rules, a goal, an initial state and a live hypermedia API instance, a PPA-implementation should be able to reach the goal. In conjunction with the assumption that the RESTdesc descriptions can be obtained through an `OPTIONS *` request, this means that only knowledge of RESTdesc, RDF and HTTP are assumed and any hypermedia API using these technologies can be used for achieving goals *without* programming.

To the best of the author's knowledge, there was no FLOSS-implementation of the PPA available. Thus, it was implemented. Python was used as the programming language, using the `rdflib` and `requests` libraries for graph manipulation/querying and sending HTTP requests, respectively. The EYE reasoner [115] is used for creating the necessary proofs. The source code for the PPA-implementation is released under the MIT licence on GitHub at `https://github.com/UdSAES/pragmatic-proof-agent`.

5.4.2 The Extended Pragmatic Proof Algorithm

The original version of the PPA does not account for user input which has requirements that only become known at run-time, such as the parameters for the instantiation of a model just added. In an RDF hypermedia API, this could manifest itself in endpoints that require RDF graphs with certain properties as input, equivalent to forms on a web page.

These requirements on input data have to be communicated at runtime, but at the same time, the PPA needs to know about the fact that input conforming to a schema will be required eventually so it is enabled to prepare complete/valid requests when they become relevant. We propose the following solution as an extension of the PPA: independent of any specific interaction, the existence of constraints on user-supplied input is communicated in the corresponding RESTdesc rule, as described earlier.

When attempting to generate the initial pre-proof, it needs to be established that the precondition can be met. Therefore, it must be communicated that the definition of the shape will be communicated as part of the interaction between service and consumer, which is expressed in the postcondition of the RESTdesc rule describing the effects of the corresponding request. As an example, consider lines 26 to 27 in listing 5.3. Their meaning is that "there is a node which is a SHACL shapes graph that has a target node". Input conforming to the shapes graph cannot be prepared yet because the shapes graph is empty. In order to proceed anyway, it is assumed that matching input will be available when needed as follows: for each shape identified in the rules, its target node (an existentially quantified variable in the RESTdesc description, `_:parameterSet` in the example) is replaced with a link to an empty file. The subject of the triple (`_:instantiationShape` in this case) is kept as a variable and the triple `_:instantiationShape sh:targetNode <file:///tmp/input_00.n3>` is added as background knowledge to the API composition problem. As soon as a response contains the definition of the shape, in which the target node is an empty blank node, the variable is replaced by the URI of the shape so that the extended PPA arrives at a triple which states that the target node of the shapes graph specified in an API response is the empty file

(`<file:///tmp/input_00.n3>` in this case).

When the PPA comes to the point where the next request to be sent includes a body and the term identifying the body is specified through a shape, the extended PPA asks its user to supply input conforming to the shape *in the previously empty file* before proceeding. The user here is a higher-level agent responsible for providing meaningful input, for example by querying a KG based on the shapes graph and then selecting exactly one query result. This step requires knowledge and algorithms which are outside the scope of the PPA and are thus excluded from the PPA-implementation. ▲

6. Applications

In the preceding chapters, the motivation for combining the modelling and simulation of technical systems with the Semantic Web and the expected benefits from this combination as suggested in the literature were outlined. In the last chapter, an in-depth look at the design of the developed M&S hypermedia API with respect to its software architecture and -engineering as well as the resource modelling was taken. This included the design of the FMI- and SMS-ontologies as well as their application to represent dynamic system models in the RDF data model. Then, necessary building blocks for dealing with the questions that govern the interaction between service provider and service consumer were identified and possible specific solutions to these questions were explained.

Now, it is demonstrated that the previously discussed selection/combination of concepts, architectures and algorithms can be *useful* in specific situations. First, an overview of all deployed implementations of concepts, architectures and algorithms is given. Then, exemplary specific uses of these software components are summarized; their observed characteristics are analysed; and possible generalizations and other uses which could benefit from the observed characteristics are outlined.

In all scenarios, it is assumed that ready-to-use models including information on what the model represents exist in the form of a FMU because the *creation* of models is outside the scope of the developed M&S hypermedia API.

In figure 6.1, an overview of the deployed services as well as their consumers is presented. Two services are deployed: obviously, an instance of the M&S hypermedia API provides access to M&S capabilities. The service interface can be viewed from three distinct perspectives: first, there is a QPF interface that allows *searching* for knowledge about models and model instances that the service instance holds. Then, there are the resources and operations that expose the actual *capabilities and data* (models, model instances, simulations, simulation results). Last, a *description* of the service is exposed in the form of RESTdesc descriptions as well as a static interface description according to the OAS.

Additionally, an instance of the `Linked Data Fragments Server` provides a QPF interface to an exemplary SE knowledge graph. The KG holds knowledge about the *context* in which the M&S resources are used. For example, it contains triples on systems represented through model instances; persons involved in the creation and use of models; data sets for validating simulation results; as well as scientific publications for which models and/or data sets were used.

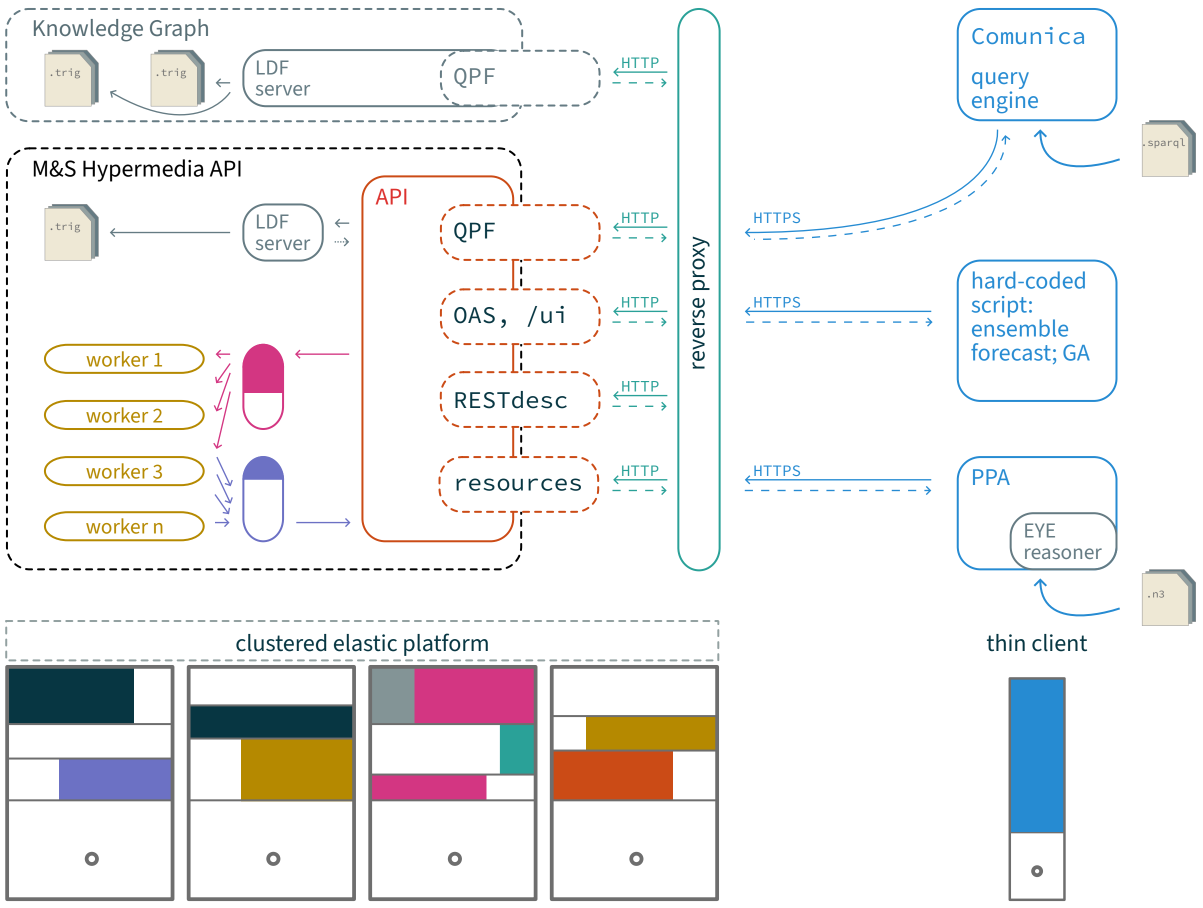

Figure 6.1: For running the scenarios that are intended to demonstrate the usefulness of the chosen approach, an instance of the M&S hypermedia API and a systems engineering knowledge graph are deployed on a Kubernetes cluster which manages several powerful servers. The consumers can run on thin clients and interact with the services through a reverse proxy, which encrypts the connection from the cluster to the users.

Three exemplary consumers use the services to implement the chosen scenarios.

- The Comunica framework [100] is used as the query engine that translates SPARQL queries to the necessary requests to one or more QPF interfaces, executes them and combines the responses to form the answer to the original SPARQL query.
- A script written in Python serves as an example of a software client that is using requests which are hard-coded at design-time against a specific API version to achieve a goal.
- In contrast, an implementation of the Pragmatic Proof Algorithm is used as an example of a generic software agent that uses the REST-desc descriptions of the API to determine whether its goal can be achieved using the available hypermedia APIs and then constructs the necessary requests based on the information contained in the resource representations obtained at run-time to realize the goal.

The services are deployed on a Kubernetes-installation as an example of a clustered elastic platform. This means that the computing resources, such as the number of central processing unit (CPU) cores that can be used by the services, can exceed the capabilities of a single machine because several physical machines are combined into a pool of resources. The details of addressing and allocating these resources are handled by Kubernetes transparently, in other words it is not necessary to account for this complexity as part of the service implementation.

6.1 Finding Relevant Models

The first group of scenarios is concerned with *finding* models that are relevant in a given situation. For example, one might want to choose models to use; to retrace the provenance of a model; or to identify models which were used in a certain context.

The data necessary to answer such queries is directly associated with the models or model instances themselves, either included in the format chosen for encoding a model, such as metadata included in a FMU, or automatically derived from the model format, such as data on the inputs and outputs of a FMU. Formally, this data is encoded in the RDF data model using the FMI-, SMS- and other ontologies upon adding models or model instances and stored by an instance of the developed M&S hypermedia API. Access to the resulting KG is provided through service's QPF interface. For now, this is the only source of data used for answering queries. Broader queries that can be answered by combining the data on models with additional context will be discussed in section 6.3.

Consider the excerpt of a KG held by an instance of the M&S hypermedia API visualized in listing 6.1. There are three models. For each model, there are some triples that encode descriptive metadata, basic provenance information, explicit statements about what a model represents, as well as triples about inputs/outputs as derived from the model. This enables queries based on the properties of the models themselves; their history and organizational context; and the role that they can or do assume in an MBSE context. Several specific examples are shown next in terms of the question in natural language, the SPARQL query that implements the question and the result of running the query against the KG shown in listing 6.1.

Models Representing Classes of Systems

Which models represent PV systems or wind turbines in general?

```
SELECT ?model ?classOfSystems WHERE {
  VALUES ?classOfSystems {
    <http://dbpedia.org/resource/Photovoltaic_system>
    <http://dbpedia.org/resource/Wind_turbine>
  }
  ?model rdf:type sms:Model .
  ?model sms:represents ?classOfSystems .
}
```

?model	?classOfSystems
<m2>	dbpedia:Photovoltaic_system
<m3>	dbpedia:Photovoltaic_system

Model Provenance

What is known about the provenance of model m3? Specifically, which entities is it derived from through which activities, and by whom?

```
SELECT ?parent ?activity ?child ?author WHERE {
  ?child prov:wasAttributedTo ?author .
  ?child prov:wasGeneratedBy ?activity .
  ?activity prov:endedAtTime ?date .

  OPTIONAL {
    ?child prov:wasDerivedFrom ?parent .
  }
}
ORDER BY ASC (?date)
```

?parent	?activity	?child	?author
<m2>	:modelExport_m3	<m3>	ppl:Moritz
<m1>	:implementation_m2	<m2>	ppl:Moritz
	:implementation_m1	<m1>	ppl:Danny

Models Linked to Publications

Are there any models which are associated with scientific publications?

```
SELECT ?model ?publication WHERE {
  ?model a sms:Model .
  ?publication a bibo:AcademicArticle .
  ?model prov:influenced ?publication .
}
```

```
@prefix dct: <http://purl.org/dc/terms/> .
@prefix fmi: <https://purl.org/fmi-ontology#> .
@prefix sms: <https://purl.org/sms-ontology#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix bibo: <http://purl.org/ontology/bibo/> .
@prefix qk: <http://qudt.org/vocab/quantitykind/> .

@prefix ppl: <http://example.com/people/> .
@prefix lang: <http://example.com/languages/> .
@prefix : <http://example.com/> .

@base <http://example.com/models/> .

<m3> a fmi:FMU, sms:Model, prov:Entity ;
  fmi:hasInput [ qk:quantitykind qk:Irradiance],
    [ qk:quantitykind qk:Temperature], [qk:quantitykind qk:Speed], ...
  fmi:hasOutput [ qk:quantitykind qk:Power ] ;
  sms:represents <http://dbpedia.org/resource/Photovoltaic_system> ;
  sms:formalismUsed lang:FMI ;
  prov:wasAttributedTo ppl:Moritz ;
  prov:wasDerivedFrom <m2> ;
  prov:wasGeneratedBy :modelExport_m3 ;
  prov:influenced <http://doi.org/10.3389/fenrg.2021.639346> ;
  ...

<m2> a sms:Model, prov:Entity ;
  sms:represents <http://dbpedia.org/resource/Photovoltaic_system> ;
  sms:formalismUsed lang:Modelica ;
  prov:wasAttributedTo ppl:Moritz ;
  prov:wasDerivedFrom <m1> ;
  prov:wasGeneratedBy :implementation_m2 ;
  ...

<m1> a sms:Model, prov:Entity ;
  sms:formalismUsed lang:Fortran ;
  prov:wasAttributedTo ppl:Danny ;
  prov:wasGeneratedBy :implementation_m1 ;
  prov:influenced <http://doi.org/10.18086/eurosun2018.02.16> ;
  ...

:modelExport_m3 a prov:Activity; prov:endedAtTime "2022-03-07" .
:implementation_m2 a prov:Activity; prov:endedAtTime "2020-12-24" .
:implementation_m1 a prov:Activity; prov:endedAtTime "2019-11-12" .

<http://doi.org/10.3389/fenrg.2021.639346> a bibo:AcademicArticle ; ...
<http://doi.org/10.18086/eurosun2018.02.16> a bibo:AcademicArticle ; ...
```

Listing 6.1: The excerpt of a knowledge graph shown includes provenance data about several models, such as their time of creation, interrelations, and responsible persons.

?model	?publication
`<m1>`	doi:10.18086/eurosun2018.02.16
`<m3>`	doi:10.3389/fenrg.2021.639346

From these exemplary queries, it is concluded that representing data about models in RDF and making them available for querying is useful for two main reasons:

- searching for entities based on their meaning and/or relations to other entities is enabled; and
- implicit knowledge is made explicit.

This is even the case when only relatively few triples such as those shown in listing 6.1 (basic metadata, basic provenance, ...) are available.

Beyond the specific examples shown, it is suspected that everyone involved in MBSE could benefit from better findability of models, especially if the creation of component models is not the focus of the work, but only a necessary step to arrive at system models used for analysing system behaviour.

Making knowledge that is typically only available implicitly (in unstructured documents or the memories of people involved) *explicit* could make a lot of sense in environments with a very large body of knowledge distributed across several organizational units; as well as for making organizations more resilient against changes in personnel and the loss of knowledge associated with it.

6.2 Simulating Models in the Cloud

The second group of scenarios concern themselves with using the developed service to execute simulations in the cloud. Two approaches are taken: initially, the necessary requests are programmed at design-time against the documentation of the service interface. The focus is on the execution of hundreds of simulations and the achieved characteristics for running them. Programming the requests at design-time is straightforward from both the conceptual and the practical perspective, meaning that it is easy to integrate into larger processes. However, the approach has drawbacks—programmers are required each time a new model needs to be used or the service interface changes.

Then, the requests are constructed at run-time based on the information provided in the resource representations—in other words, hypermedia is used as the engine of application state.

6.2.1 Requests Programmed at Design-Time

The content of this subsection was presented at the Modelica conference 2021 [93].

To demonstrate that a cloud-native, open-source implementation of the MSaaS-concept can be useful, the generation of an ensemble forecast for the power generated by a PV system as well as the component selection for a thermoelectric circuit by means of a genetic algorithm (GA) were implemented.

For both scenarios, it is assumed that the model, its parameters, inputs and simulation settings as well as an instance of the service are given. The HTTP requests necessary are programmed manually, before running the scenario. In case sequences of requests are necessary to implement a functionality, for example `POST` ing a representation of a simulation and then repeatedly polling its status to eventually retrieve the simulation results using `GET` -requests, the requests are executed in order. Requests and requests sequences that are independent of each other are executed in parallel whenever possible—for example, the request sequences for simulating two different model instances are executed concurrently.

Ensemble Forecast

The PV system model calculates the power and energy generated by a PV system subject to the environmental conditions at the site. It comprises the calculation of the generated power from the global irradiance in the plane of array (POA), the air temperature and the wind speed; as well as the transformation of the horizontal irradiance (which is typically provided by numerical weather predictions (NWPs)) into the POA, including the calculation of the sun's position relative to the system's position, which is required for this conversion.

Details can be found in a publication on the forecast quality achieved using an instance of the model [97, section 3.1].

The motivation for generating an ensemble forecast is that plotting a single trajectory of values over time does not represent the uncertainties inherent to M&S . It implies that the solution was exact as it neither shows any information about the uncertainty of the result, nor about its confidence levels. As a consequence, the plots can be misleading, which frequently results in misinterpretations [105]. An ensemble forecast, in other words plotting several simulation results for which one or several of model, parameterization, input data and simulation settings are slightly varied, is one way to more accurately communicate the meaning of simulation results, including their uncertain nature.

Uncertainties arise from the chosen abstraction and its formalization; parameter values; input data; numerical approximation; et cetera.

Forecasting the power generation of a PV system with the different members of an ensemble NWP as input represents a case where the same model instance is simulated many times with different input trajectories.

Compared to simulating a FMU using `fmpy` directly, the use of the MSaaS-implementation reduces the complexity for executing simulations from interfacing specialized software in a programming language locally to sending HTTP requests in the correct order. This enhances the *accessibility* of the functionality "executing simulations using FMUs", and thus facilitates *reuse*, *repeatability* of simulation studies and *composition* of M&S resources. For example, the PV forecast could be part of a distributed, recurring process which provides the necessary information for supervising and controlling smart grids. In this scenario, a utility company would require forecasts for many systems every few hours. Using a MSaaS-instance instead of creating the forecasts locally would mean that the utility company does not need to invest in hardware, software and personnel, which

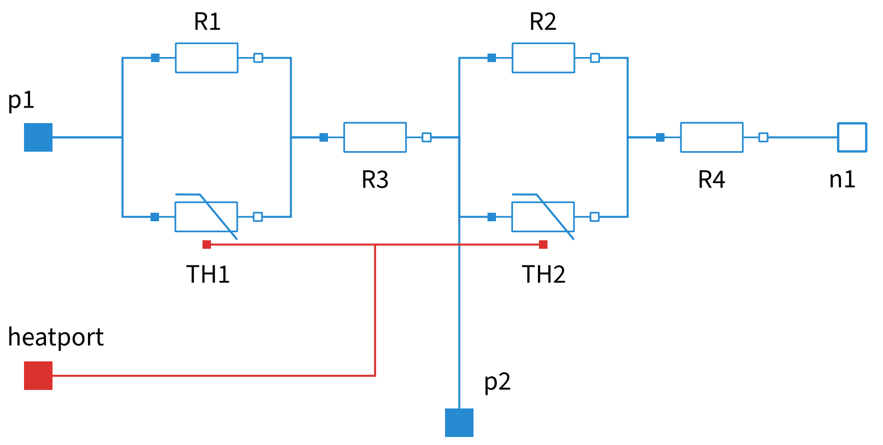

Figure 6.2: A thermistor bridge can be used to compensate for changes in the temperature at which a circuit operates.

is likely outside their core business model. Instead, they could pay but for exactly the simulations that they require; benefit from the expertise of the service provider and focus on their core task.

Component Selection

The computational search for the optimal sizing of components in a system model represents a case in which many model instances need to be simulated with the same initial conditions and inputs. For example, consider the temperature-dependent electrical circuit shown in figure 6.2. The purpose of the simulations is to find a good set of component values for the resistors and thermistors, given a desired voltage at the junction p2 over the temperature range from −10 °C to 60 °C.

Assuming that there are nine possible values for both the resistance at reference temperature and the temperature coefficient B of the two thermistors, and assuming that each resistor can take one of the 70 values of the E24-series between 300 Ω and 220 kΩ, there are $70^4 \cdot 9^2 \cdot 9^2 = 157\,529\,610\,000$ possible combinations. If it took a CPU time of 0.05 s to simulate one variant, testing every permutation would require roughly 250 years of computing time.

An alternative approach is to use a genetic algorithm (GA) to find a good solution without trying *every* permutation, as suggested for this problem in an article on edn.com [27].

▾ [93, section 4.2]

A genetic algorithm mimics the 'survival of the fittest' principle—see [77] for an introduction.

Each possible combination of component values is seen as an individual. The fitness of an individual is evaluated by how close the voltage at p2 matches the desired voltage over the relevant temperature range, for example in terms of the root-mean-square error (RMSE). Because the determination of the fitness of an individual is independent of other individuals, the fitness values for an entire generation can be evaluated in parallel. After the fitness of each individual in a population is determined, the best results are recorded and the next generation is created by cross-over and mutation (subject to user-defined probabilities). The algorithm is terminated

by setting a threshold for either an acceptable fitness value or a fixed number of generations.
This example represents a situation where different model instances are simulated with the same input. For finding good solutions, a few hundred simulations are likely required, many of which can be executed in parallel given a sufficiently high number of worker instances. Using a MSaaS-instance deployed on a clustered elastic platform instead of running the GA locally becomes really beneficial iff the number of individuals in a generation is higher than the number of CPUs available locally.
Because the example is merely intended to serve as a proof of concept, no detailed analysis of the performance was carried out—neither with respect to the speed-up achieved, nor with respect to the overhead introduced by the additional software layers and the exchange of data over the network.

For implementation of the GA, the Distributed Evolutionary Algorithms in Python (DEAP) framework [34] was used. A tiny Modelica package containing the necessary models and an example ready to be simulated in Dymola is included in the `simaas-demo`-repository[1]. ▲

In conclusion, the scenarios are intended to highlight two aspects of providing M&S-functionality as a cloud-native service: from a user's perspective, the requirements on the environment that the user has to provide and the overall complexity to achieve a goal are decreased. The technical prowess necessary to run the simulations is hidden from the user and the reduction of the interface complexity facilitates the use of simulations at the expense of less fine-tuned control over the simulations themselves.

From a provider's perspective, the applications show that the implementation realizes some of the defining characteristics of cloud computing. Specifically, the functionality is offered to consumers as an on-demand self-service without apparent limitations on the available computing power. Many requests can be received in a short amount of time and still worked on in a robust manner due to the asynchronous implementation of the API component and the distribution of computing intensive tasks to available workers via a queue. Deploying several worker instances enables the execution of mutually independent simulations *in parallel*. When deployed on a clustered elastic platform, the worker instances can be added or removed automatically in response to the current demand.

Due to the achieved characteristics, it is suspected that using a cloud-native implementation of the MSaaS-concept is beneficial in situations where many simulations need to be evaluated as a whole and it is desirable to keep the time they take to complete low. For example, tens of simulations might be necessary for displaying ensemble forecasts on websites, for which it is important to keep the time it takes to create them low in order to not lose the user's attention. Hundreds of simulations may be necessary for parameter

[1] https://github.com/UdSAES/simaas-demo

fitting or sensitivity analyses; as well as for choosing a configuration of available subcomponents that fit a customer's requirements best.

6.2.2 Requests Constructed at Run-Time

The core hypothesis of using an implementation of the PPA as the consumer of a hypermedia API is that no programming *specific to the API used* is necessary. Instead, the goal to be achieved is formulated as an N3 rule and initial knowledge as well as (optional) background knowledge are defined. Then, the PPA determines whether the goal is achievable given the available hypermedia APIs based on the RESTdesc descriptions they provide. If the goal is achievable, the correct sequence and -form of the requests to be sent is figured out based on the hypermedia representations of resources retrieved from the APIs. In case the API requires input according to a specific shape which *only becomes known at run-time*, the extended version of the PPA identifies these shapes and asks a higher-level user to supply the matching input when required.

Two examples were implemented to demonstrate the combined functionality of hypermedia API and the Pragmatic Proof Algorithm. Initially, the image-resizing example used by Verborgh et al. to explain RESTdesc and the PPA [119, section 4] was repeated to verify that our PPA-implementation functions correctly. Then, addition, instantiation and simulation of a FMU and the retrieval of the simulation results were specified as the goal for the PPA-implementation. In contrast to the first example, here the agent is required to provide input matching a shape which is only communicated at run-time as part of the interaction.

Three software components are involved to implement the scenarios: the M&S hypermedia API (resources exposing capabilities and RESTdesc descriptions); a PPA-implementation including the necessary instance of the Euler Yet another proof Engine (EYE) reasoner acting as the service consumer; and a higher-level user which supplies correctly shaped input to the PPA-implementation at run-time if and when required. Here, a simple script which hard-codes the data for the chosen example is used because a more sophisticated solution is considered out of scope.

Resizing an Image

Consider a hypermedia API that generates a low-resolution thumbnail of an image. There are two requests necessary to obtain the thumbnail, which is explained using two RESTdesc rules. First, an image of type `dbpedia:Image` can be uploaded in the body of a `POST`-request to `/images`. As the reaction to a successful upload, the API provides a link to the thumbnail. Second, the thumbnail itself can be downloaded through a `GET` request to this link, and the original image will be linked to the thumbnail via the `dbpedia-owl:thumbnail`-relation.

Now, given an instance of the hypermedia API; the initial knowledge `<example.png> a dbpedia:Image`; and the goal state

```
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.

{ <example.png> dbpedia-owl:thumbnail ?thumbnail. }
=>
{ <example.png> dbpedia-owl:thumbnail ?thumbnail. }
```

in the form of an N3 filter rule, the PPA can infer the necessary requests to achieve this goal and execute them in the correct order. Additionally, the example was extended such that the paths at which the resources are exposed can be changed, for example to the German translation `/bilder/<id>/miniaturbild`. Essentially, this results in three (English, German, French) versions of the hypermedia API. Requests hardcoded against one of the versions at design-time will fail with the other versions, whereas the PPA works without modification for every version because it realizes the HATEOAS principle, constructing the requests at run-time based on the information provided during the service interaction.

▼ [94, section 5.1]

Simulating a Model

Assume that a forecast for the power produced by a specific PV system for the next day is required. For example, the forecast might be used for optimizing the energy consumption in a microgrid with the objective of maximizing the local (own) use of the generated energy. To create the desired forecast, access to a model representing PV systems with an adequate accuracy, and the means to simulate it are required. If a user lacks one or both, a service that provides them must be used, such as the M&S hypermedia API. The service consumer then needs to provide parameters (panel area, panel orientation, system location, ...); external conditions in the form of input time series (irradiance, temperature, wind speed, ...); and simulation settings (start and stop time, temporal resolution, ...). Of these consumer inputs, some are required while others are optional because sensible default values can be used. Provided that the consumer has the necessary data, the correct sequence of requests needs to be identified, and the input data likely has to be reshaped such that it matches the expectations of the API.

In terms of the conceptual resources exposed by the API, the model needs to be uploaded and then instantiated before the simulation using this model instance can be specified and, eventually, the simulation results can be retrieved. This is expressed through the goal state g (listing 6.2). The triple `<file:///tmp/model.fmu> a fmi:FMU` is the initial knowledge H.

With this and a live M&S hypermedia API-instance, the PPA-implementation can be started. As the first step, the RESTdesc descriptions are downloaded through an `OPTIONS` request to `*`. Then, it is checked whether any shapes are expected to specify requirements on data to be uploaded. In this case, two shapes are found: one for instantiating a model and one for specifying the simulation to be run. The expectations are added to the API composition problem as

```
@prefix sms: <https://purl.org/sms-ontology#> .

{
    ?modelInstance sms:instanceOf ?model .
    ?simulation sms:simulates ?modelInstance .
    ?simulationResult sms:resultOf ?simulation .
}
=>
{
    ?modelInstance sms:instanceOf ?model .
    ?simulation sms:simulates ?modelInstance .
    ?simulationResult sms:resultOf ?simulation .
} .
```

Listing 6.2: An N3 rule expresses that the results of a simulation of an instance of a model are the goal to be achieved by the Pragmatic Proof Algorithm.

background knowledge.

Next, the PPA-implementation attempts to create the initial pre-proof using the EYE reasoner. A proof is found, therefore the goal is achievable. The request in the RESTdesc rule for adding the FMU to the API instance through a `POST` request to `/models` is fully specified (no unknown universally quantified variables), so it is executed. The PPA learns that `</models/6157f34f> rdf:type sms:Model` through the response, so the precondition in the first rule in listing 5.3 is now met and, as a consequence, the request fully specified. The request is executed next, after it was confirmed that the initial `POST` request contributed to achieving the goal by generating a post-proof and recognizing that the number of remaining requests decreased compared to the pre-proof.

▲

The response to the `GET` request to `/models/6157f34f` contains the definition of `/models/6157f34f#shapes-instantiation` for creating an instance of a model.

Through the extension of the PPA, the background knowledge is now updated to contain the triple

```
</models/6157f34f#shapes-instantiation>
sh:targetNode
<file:///tmp/input_00.n3> .
```

Before executing the request `POST /models/6157f34f/instances`, which is now fully specified because the triple listed above is known, the higher-level user of the PPA-implementation is asked to supply the input data to be sent as the body of this request inside the file `/tmp/input_00.n3`. The contents of the file are then sent as the body of the request.

The remaining requests are identified and executed in the same manner until the simulation result is retrieved, and a proof confirms that the goal has been achieved.

Successfully running the examples using the same, unaltered implementation of the PPA demonstrates that the developed M&S hypermedia API is indeed usable by generic software agents. Both examples are included in the repository `https://github.com/UdSAES/pragmatic-proof-agent`.

Only few assumptions are made, which are considered reasonable for a proof of concept implementation: the URLs of the available hypermedia APIs must be supplied; the hypermedia APIs must transfer the RESTdesc descriptions as the response to a `OPTIONS`-request to `*`; and the HTTP-ontology[1] must be used to communicate the existence and form of HTTP requests that allow transitioning between resource states inside the RESTdesc descriptions.

Previously, it was necessary to program a client for every model to be used with a non-hypermedia implementation of the MSaaS-concept. Now, one client can use all capabilities of all hypermedia APIs that meet the assumptions outlined above. This includes combinations of different hypermedia APIs, in other words automatic mash-ups. A declarative formulation of goals to be achieved using services becomes possible, which makes the manual implementation of requests obsolete. Moreover, clients become robust against changes in the service interface because they no longer rely on information hard-coded at design-time.

The value of these gained capabilities is expected to manifest itself in distributed systems of systems which include many entities such as sensors, models and actuators, likely subject to frequent reconfiguration.

For example, Arndt et al. [2] describe such a system in the context of patient care in hospitals and outline the use of N3 rules very similar to RESTdesc that describe the effect of possible sensor queries; ontological knowledge about the context; individuals instantiating classes defined in the ontologies; and generating proofs via reasoning to identify sensors and threshold values relevant to a certain goal. The authors explicitly state that the rules to describe the effect of sensor queries can be combined with RESTdesc rules that describe the effect of hypermedia APIs [2, section 5].

6.3 A Systems Engineering Knowledge Graph

Hypermedia representations that contain links of one piece of data to other pieces of data are a core mechanism of the Web: the existence of links enables both using the representations themselves for advancing the application state and the *serendipity*, in other words "the faculty or phenomenon of finding valuable or agreeable things not sought for" [63], which has become characteristic for browsing the Web. Both are seen as core reasons for its success.

Consumers usually depend on links *provided by the services* they use; which is why the FAIR-principles recommend that "(meta)data are richly described with a plurality of accurate and relevant attributes" including a "clear and accessible data usage license" and "detailed provenance" (principles R1, R1.1, R1.2; see table 2.1). In the context of the Semantic Web, the data model of choice to be used by software agents to consume such

[1] `http://www.w3.org/2011/http#`

Prefix	URL	Purpose/Topic
bibo	http://purl.org/ontology/bibo/	Bibliographic details
dcat	http://www.w3.org/ns/dcat#	Description of data sets
dct	http://purl.org/dc/terms/	Basic metadata
foaf	http://xmlns.com/foaf/0.1/	Persons and their relationships
geo	http://www.w3.org/2003/01/geo/wgs84_pos#	Encoding locations
prov	http://www.w3.org/ns/prov#	Encoding of provenance data
qudt	http://qudt.org/schema/qudt/	Encoding of physical quantities
unit	http://qudt.org/vocab/unit/	Encoding of units
sosa	http://www.w3.org/ns/sosa/	Sensors, Observations, Samples, Actuators
spdx	http://spdx.org/rdf/terms#	Licensing information
time	http://www.w3.org/2006/time#	Encoding time data

Table 6.1: There are many ontologies that can be used to represent data in a systems engineering context in an RDF graph; this table shows those used in the exemplary KG created as part of this thesis.

data is RDF; in its role as a foundational building block of the Semantic Web it can be seen as the lowest common denominator that participants of a conversation have to agree on in order to enable the exchange of data. Consequently, it means that the data to which links are given should also be available in RDF to allow the traversal and exploration of a domain of interest.

To demonstrate that the availability of triples about a MBSE context allows answering meaningful questions that could not be answered programmatically before, an exemplary systems engineering knowledge graph was populated and then used to answer SPARQL queries in conjunction with the QPF endpoint of the M&S hypermedia API.

The content of the KG is themed around the PV performance forecast example introduced earlier: it includes power measurements taken at two PV systems and the systems' properties; data about weather stations in the vicinity of these systems, including weather measurements and -forecasts; statements about people and organizations involved in the systems' operation and use; as well as bibliographic information on related scientific publications and the ORCiD-records of involved persons. Moreover, some statements were manually added to tie different data sets together, such as `owl:sameAs` -statements for entities that were identified differently in different data sources, or more details on the data's context. For representing all the above in RDF, established ontologies could be used, which are listed in table 6.1.

In figure 6.3, the pipeline used to populate the KG from the different data sets is visualized. The steps taken are based on Ruben Verborgh's pipeline for creating an RDF-representation of his personal website [111]. To begin, for data that exists in semi-structured form such as CSV or JSON, the `RML`[1] mapping language [26] was used to define how elements of this data shall be represented as RDF triples. The actual translation is then executed by an instance of `RMLMapper`[2].

Some services directly provide RDF representations of their resources through content negotiation. For example, when the `Accept` -header is

[1] https://rml.io/
[2] https://github.com/RMLio/rmlmapper-java

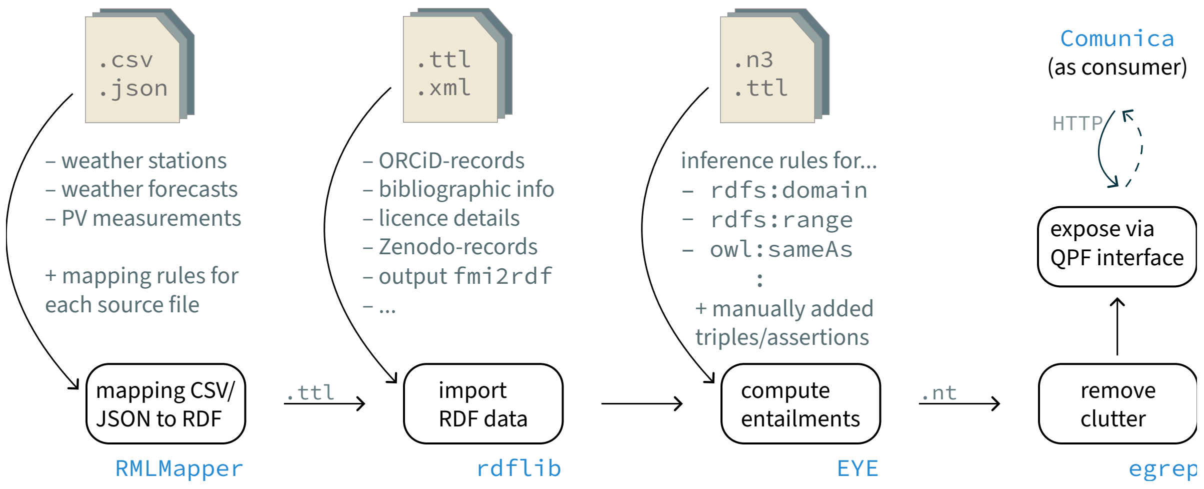

Figure 6.3: Data from semi-structured sources is transformed to RDF via RML-mappings; some sources directly provide triples. After import using `rdflib`, the `EYE` reasoner applies rules to infer additional triples. Of these, some that do not add relevant information are removed again using the command-line tool `egrep` before storing all triples and exposing them through a QPF interface such that agents can query the data using `Comunica`.

set to `text/turtle` instead of `text/html`, sending a `GET`-request to `https://doi.org/10.3389/fenrg.2021.639346` results in a document containing an RDF-representation of the article's bibliographic details serialized in the Turtle language instead of the HTML version rendered as a website by browsers[1]. The dblp computer science bibliography and ORCiD are other examples of services that provide RDF representations of the data that they hold.

Next, the EYE reasoner is used to compute entailments, in other words to explicitly list all triples that can be inferred from applying knowledge encoded in the ontologies used to the triples that are generated directly. Thereby, the KG explicitly contains most of the links that might be required when answering queries. This decision to collect all data in one triple store and to perform reasoning during import does not reflect the distributed nature of the Semantic Web, but it simplifies the realization of the scenarios and was thus chosen for implementation. All triples are stored in a file which acts as the triple store. The triples are made accessible through a QPF server; requests to it are generated from SPARQL queries through the `Comunica` framework.

The queries used as examples were chosen such that they represent three different aspects of using a KG: first, there are queries for discovering the content of the KG and finding specific information. Second, some queries might help with deciding to which extent a resource can be trusted. Third, information on the use of models such as the prerequisites for simulating them (required parameters, input trajectories) can be obtained from the KG. As in section 6.1, they are discussed in terms of their formulation in natural language and SPARQL; the results when executed against the KG of which an excerpt is shown in listing 6.3; and remarks if applicable. The difference to section 6.1 is that now data on a much broader context, beyond the data known to an instance of the M&S hypermedia API, is available.

[1] `https://citation.crosscite.org/docs.html`

```
@prefix var: </models/6157f34f/variables#> .
@base <http://example.com> .

</agents/swsls> a prov:Agent, prov:Organization ;
  rdfs:label "Stadtwerke Saarlouis GmbH"@de ;
  foaf:homepage <https://www.swsls.de> ;
  prov:influenced </activities/provide-data-pv-swsls>, </systems/swsls-sw>,
    </systems/swsls-sw/behaviour>; ... .

</activities/provide-data-pv-swsls> a prov:Activity ;
  prov:generated </systems/swsls-sw/behaviour>, ... ;
  prov:wasAssociatedWith </agents/swsls> ; ... .

</systems/swsls-sw> a geo:SpatialThing, prov:Entity, sosa:Platform ;
  dct:title "Photovoltaikanlage Stadtwerke Saarlouis"@de ;
  geo:lat "49.319986" ; geo:long "6.746344" ; geo:alt "181" ;
  prov:wasAttributedTo </agents/swsls> ;
  foaf:based_near </weather-stations/10704>; ... .

</weather-stations/10704> a geo:SpatialThing, prov:Entity, sosa:Platform ;
  geo:lat "49.264019" ; geo:long "6.686793" ; geo:alt "363" ;
  dct:title "Berus"@de ; foaf:based_near </systems/swsls-sw> ;
  sosa:hosts </weather-stations/10704/sensors/t_2m>,
    </weather-stations/10704/sensors/ws_10m>, ... ;

</systems/swsls-th/behaviour> a dcat:Dataset, prov:Collection, prov:Entity ;
  dct:date "2019-05-28"^^xsd:date ;
  dct:publisher </agents/swsls> ;
  prov:wasInfluencedBy </activities/provide-data-pv-swsls>, </agents/swsls> ;
  dcat:temporalResolution "PT15M"^^xsd:duration ;
  sosa:isFeatureOfInterestOf [ a prov:Activity, sosa:Observation;
    sosa:hasResult [
      a qudt:QuantityValue; qudt:numericValue 600.1; qudt:unit unit:W
    ]; sosa:phenomenonTime [ a time:Instant, time:TemporalEntity;
      time:inXSDDateTimeStamp "2019-03-15T06:45:00+0000"^^xsd:dateTimeStamp
  ]], [...]; ... .

</models/6157f34f> a fmi:FMU, sms:Model, sms:Virtual ;
  fmi:modelName "PhotoVoltaicPowerPlantFMU" ;
  fmi:hasParameter var:latitude, var:longitude, var:elevation,
    var:environmentAlbedo, var:panelArea, var:panelAzimuth, ... ;
  fmi:hasInput var:temperature, var:windSpeed,
    var:diffuseHorizontalIrradiance, var:directHorizontalIrradiance;
  fmi:hasOutput var:angleOfIncidence, var:angleOfSunAboveHorizon,
    var:powerDC, var:totalEnergyDC ;
  prov:wasAttributedTo <https://orcid.org/0000-0002-4006-8582> ;
  spdx:declaredLicense <http://spdx.org/licenses/MIT> ; ... .

var:environmentAlbedo a fmi:Parameter, fmi:ScalarVariable,
  sms:ModelParameter, sms:ModelParameterNonFree, sms:UserInput, ... ;
  dct:description "Albedo for estimation of irradiance by reflection" ;
  fmi:max 1.0 ; fmi:min 0.0 ; sms:isParameterOf </models/6157f34f> .

<https://orcid.org/0000-0002-4006-8582> a dct:Agent, geo:SpatialThing,
  prov:Agent, prov:Person, foaf:Agent, foaf:Person ;
  foaf:name "Moritz Stüber" ;
  owl:sameAs <http://id.crossref.org/contributor/moritz-stuber-aq5hb6p5uld>,
      </agents/moritz-stueber>, <https://orcid.org/0000-0002-4006-8582> ;
  foaf:publications <https://orcid.org/0000-0002-4006-8582#workspace-works> .
```

Listing 6.3: Excerpt of an SE knowledge graph, serialized using the Turtle language. For readability, most prefix definitions were omitted (see table 6.1); and URLs were shortened/made relative.

Content-specific Queries

Which datasets are available and what basic metadata do they provide?

```
CONSTRUCT {
  ?dataset rdf:type dcat:Dataset ;
           dct:title ?title ;
           dct:type ?type ;
           ... .
}
WHERE {
  ?dataset rdf:type dcat:Dataset.
  OPTIONAL {?dataset dct:title ?title}
  OPTIONAL {?dataset dct:type ?type}
  OPTIONAL {...}
}
```

```
<doi:10.5281/zenodo.4392849> a dcat:Dataset ;
  dct:title "UdSAES/pv-systems: v0.9.0" ;
  dct:creator <https://orcid.org/0000-0002-4006-8582> ;
  dct:type <http://purl.org/dc/dcmitype/Software> ;
  dct:issued "2020-12-24"^^xsd:date;
  dct:publisher </agents/zenodo> ;
  dcat:keyword "Modelica", "PV Modeling" .

</systems/swsls-sw/behaviour> a dcat:Dataset ;
  dct:issued "2019-05-28"^^xsd:date ;
  dct:publisher </agents/swsls> .

...
```

Which sensors are available at the weather station based near the PV systems operated by SWSLS?

The abbreviation SWSLS stands for “Stadtwerke Saarlouis”, a local utility company.

```
SELECT DISTINCT ?station ?sensor
WHERE {
  </agents/swsls> prov:influenced ?system.
  ?system rdf:type sosa:Platform.
  ?system foaf:based_near ?station.
  ?station rdf:type sosa:Platform.
  ?station sosa:hosts ?sensor.
  ?sensor rdf:type sosa:Sensor.
}
```

The relation `foaf:based_near` was added manually, as part of the custom triples used to meaningfully combine the data from different sources.

?station	?sensor
</weather-stations/10704>	</weather-stations/10704/sensors/t_2m>
</weather-stations/10704>	</weather-stations/10704/sensors/ws_10m>
...	...

Facilitating Trust

Which activities by SWSLS that generated a data set are known to have resulted in the publication of a journal article?

```
SELECT ?activity ?dataset ?doi ?journal_title
WHERE {
  ?activity rdf:type prov:Activity.
  ?activity prov:wasAssociatedWith </agents/swsls>.
  ?activity prov:generated ?dataset.
  ?dataset prov:influenced ?publication.
  ?publication bibo:doi ?doi.
  ?publication dct:isPartOf ?journal.
  ?journal rdf:type bibo:Journal.
  ?journal dct:title ?journal_title
}
```

...resulting in: `</activities/provide-data-pv-swsls>` generated the data set `</systems/swsls-sw/behaviour>`, which influenced the article `<doi:10.3389/fenrg.2021.639346>`, published in the journal "Frontiers in Energy Research".

Prerequisites for Using Models

What are the parameters of my model?

```
SELECT ?model ?description WHERE {
  ?model rdf:type fmi:FMU .
  ?model fmi:hasParameter ?parameter .
  ?parameter dct:description ?description .
}
```

Note that this first formulation resolves without reasoning as the relevant triples are generated by `fmi2rdf`; whereas the query formulation below (like all the other queries in this section) relies on triples added through reasoning to resolve.

```
SELECT ?model ?description WHERE {
  ?model rdf:type sms:Model .
  ?model sms:hasParameter ?parameter .
  ?parameter dct:description ?description .
}
```

?model	?description
<m1>	Latitude in decimal degrees
<m1>	Longitude in decimal degrees
<m1>	Overall surface area of all panels (combined)
<m1>	Surface tilt in degree (Horizontal equals 0°, vertical equals 90°)
...	...

For which entities in the KG do we know their location?

```
SELECT ?thing ?latitude ?longitude ?altitude
WHERE {
  ?thing rdf:type geo:SpatialThing.
  ?thing geo:lat ?latitude.
  ?thing geo:long ?longitude.
  OPTIONAL {?thing geo:alt ?altitude}.
}
```

?thing	?latitude	?longitude	?altitude
</weather-stations/10704>	49.264019	6.686793	363
</systems/swsls-sw>	49.319986	6.746344	181
...	...	...	...

In conclusion, an RDF-based KG containing statements about systems, models and related entities including their interconnections can make implicit knowledge accessible. Previously unconnected, but related entities can be linked explicitly and entities in the real or envisioned world can be mapped to their virtual counterparts and vice versa. Thereby, agents are enabled to explore the world under consideration and to draw conclusions from known facts, in other words to apply the knowledge encoded in the ontologies to the individuals relevant in a given setting via reasoning. Meaningful knowledge queries, including such that may help with fostering trust in resources, can be answered algorithmically which would have not been possible before.

7. Discussion and Outlook

At the beginning of this thesis, the question “Can M&S benefit from Semantic Web concepts?” was asked. After discussing fundamental concepts; reviewing related work; stating thesis concept and -hypotheses; and presenting realization and applications, this chapter attempts to provide an answer to this question.

To do so, it is first evaluated whether the expectations set forth in the hypotheses could be realized. Then, opportunities for applications and further research based on the work presented in this thesis are outlined.

7.1 Evaluation of Hypotheses and Research Questions

For each of the main hypotheses H1–H4, the technical contributions are discussed first, as their implementation is a prerequisite for validating or falsifying the main hypotheses.

New Ontologies and the `fmi2rdf`-Parser (H1.1)

The FMI-ontology captures concepts and roles defined in the FMI standard; the `fmi2rdf`-parser makes use of these concepts and roles to create representations of FMUs in RDF. The representations contain information about the models inputs, outputs, and parameters including their respective types, units and associated limitations. Moreover, the requirements for instantiating a model as well as simulating a model instance are captured in the form of SHACL shape graphs.

The FMI-ontology was used as part of the hypermedia API and to build exemplary KGs, using the `fmi2rdf`-parser to generate the representations. Using `fmi2rdf` is faster than a manual process, and it is less error-prone. However, there are many opportunities for refining the proof-of-concept implementation. First, the test coverage should be extended both with respect to the competency questions used and by testing on more FMUs. Second, details of the implementation such as type casting for enumerations and the generation of shapes graphs should be revised, and data provided by the creator of the FMU as vendor annotations should be integrated. Third, the FMI-ontology and `fmi2rdf` should be updated to version 3.x of the FMI standard. Because the FMI-ontology is a methodological ontology, ontology and parser have a high potential for adoption in the community.

In contrast, the SMS-ontology is a referential ontology. Its potential for uses other than referring to (parts of) models, independent of their modelling language, and indicating which classes of systems a model represents, is yet to be explored. Interesting UCs are expected, such as ontology-driven modelling; however, reasoning under SNAF is likely necessary.

Both ontologies are given a persistent URL (pURL) to give them a persistent identifier and -locator, and it is attempted to realize best practices for developing FAIR ontologies (demanded by FAIR principle I2). Ontologies and parser are published under the MIT licence on GitHub; detailed README-documents give an overview on scope and usage. Some unit tests allow for regression testing and further document the software. Consequently, H1.1 is seen as validated.

Representation of FMUs in RDF (H1)

To validate H1, two KGs were built and then used to answer semantic queries about their content. First, each instance of the M&S hypermedia API collects data about its models and model instances and exposes it through a QPF interface. Second, a SE KG themed around PV forecasting was created by collecting data from various sources; transforming them to RDF; and employing a reasoner to derive additional triples that make knowledge encoded in the concept expressions and -inclusions of the ontologies used explicit. For both, semantic queries were formulated, encoded in SPARQL and run against the KGs as shown in section 6.1 and section 6.3, respectively.

The successful resolution of the queries shows that representing FMUs and related data in RDF enables advanced querying which, to the best of my knowledge, was not possible using open-source components before. This validates H1. Using RDF provides one way of amending models with knowledge that is formally encoded (both by humans and as part of software processes). Importantly, RDF and the related tooling are both well-founded in theory (section 2.2.2) and established in practice, facilitating both the implementation of software and its integration in a larger context.

Cloud-native Implementation of the MSaaS-Concept Based on FMI (H2.1)

Essentially, two variants of a service making M&S entities and capabilities available via the internet were realized as part of this work. Both expose the same functionality. Specifically, models of the dynamic behaviour of systems, exported as FMU as a stand-alone executable model format, are supported. In other words, this means that causal MIMO blocks that are integrated with a solver can be used with the developed software.

The two variants of the service differ in the characteristics of the service interface and their degree of novelty. From a technical perspective, this manifests itself in resource representations with different expressivity and in different ways of communicating the effects that requests have. However, the underlying implementation, except for the generation of resource representations, is the same for both variants.

The first version of the service is based on REST, but does not imple-

Characteristic	Architecture & Implementation	Deployment & Operations
Broad network access	as a Service in the (Semantic) Web; 12factor app	automated build, release, run-pipeline (container-based)
On-demand self-service	as a Service in the (Semantic) Web; 12factor app	automated build, release, run-pipeline (container-based)
Resource pooling	microservices; state only stored in API	containerization; infrastructure as code
Rapid elasticity	decoupling API/worker; state only stored in API; caching; 12factor app; async/await	containerization; deployment on clustered elastic platform
Measured service	logging; ~~authentication~~; ~~usage metrics~~	log collection; ~~monitoring~~; ~~enforcing usage policies~~; ~~billing based on metrics~~

Table 7.1: Realizing the defining characteristics of cloud computing requires measures at the level of the software's architecture; its implementation; and its deployment. For the developed software, all characteristics except "measured service" were realized; measures yet to be implemented are marked ~~accordingly~~.

ment the HATEOAS constraint. It uses JSON for its resource representations, and documents the service interface according to the OAS. Technically, it is similar to what others have done before. Within the Modelica community, it distinguishes itself from other lines of work through the focus on complete system models instead of more technical interfaces to the underlying FMUs, and also because it does not provide a UI since it is intended as a building block for applications and not an end-user facing application itself. This variant is new and useful [93, section 6] and represents the current best practice for providing APIs in the Web.

The second version of the service additionally realizes the HATEOAS constraint and is therefore denoted "M&S hypermedia API". It uses serializations of the RDF data model for the resource representations and explains the effect of state-changing operations through machine-actionable RESTdesc descriptions. To the best of my knowledge, it is the first realization of the MSaaS idea as a hypermedia API which is based on FMI *as well as* made available openly and documented in detail [93, 94, this thesis].

Both variants improve the accessibility of M&S entities and capabilities by providing them *as a service*. For the service provider, this has the main advantages of increased outreach and insight into the usage of their assets while protecting their intellectual property (IP), at the cost of the effort for DevOps and ensuring dependability and security. The service consumers benefit from better accessibility; an interface tailored to their needs (in this case, whole system models instead of FMI functions); the ability to embed simulations into larger processes; virtually unlimited computing power; and no installation/software requirements. In exchange, the consumers have to accept a dependency on the service provider and, consequently, less control over functionality and availability.

Four of the five defining characteristics for CNAs were realized in the developed service (table 7.1). The key aspects to achieve this are...

- to structure the software as a composition of microservices and to decouple the API from the worker components;

- to adhere to the best practices for building SaaS known as the "twelve-factor app"; and
- to use containers as deployment units.

There are two reasons why the "measured service"-characteristic was not implemented. First, it is not required for demonstrating the functionality of the service, but would require both the definition and implementation of appropriate *usage metrics* as the basis for monitoring service instances and billing service consumers and an authentication procedure as the basis for enforcing usage policies and billing. Second, such features relating to reliability, observability and security, are mostly independent of the application logic. Therefore, they should likely be implemented using sidecar containers that form a *service mesh* [68] and not as part of the service implementation itself.

▼ [94, section 7]

Regarding support for tracing and managing requirements, changes, configurations et cetera, the immutability of the exposed resources (none of models, instances, simulations or simulation results can be modified after their creation) facilitates their integration into a management system at a higher level. If necessary, relevant triples could be added to the resource representations, but this is currently not implemented.

▲

There are some implementation details that ought to be addressed before running the service in a production environment. Most importantly, these are the lack of an explicit threat model/consideration of security and dependability aspects; the lack of a mechanism for enforcing usage policies; and a possible scalability bottleneck because the API component is not designed to scale by deploying more instances. Still, H2.1 is seen as valid and idea and existence of the software were communicated in the Modelica community [93].

▼ [94, section 6.1.3]

Support for Loose Coupling (H2.2; Q3)

The result of analysing the REST-based HTTP-API and the M&S hypermedia API with respect to coupling according to the facets suggested by Pautasso and Wilde [76] (see section 2.3.1) is visualized in figure 7.1. The detailed assessment can be found in table A.4 in appendix A.

Note that the axis is reversed compared to the visualizations in [76] for better consistency within this document.

The same value is assigned for both variants for the facets discovery (referral), identification/naming (global), platform-dependency (independent), interaction (synchronous), granularity (depends on implementation) and state (stateless). This is the consequence of basing the design of both alternatives on REST, specifically the use of URLs, server/client interaction via HTTP and the exchange of stateless messages.

The only facets for which the hypermedia API is not classified as supporting loose coupling are granularity and interaction. Granularity is defined as "the design trade-off between the number of interactions that are required to provide certain functionality to service a large client community, and the complexity of the data parameters (or operation signatures) to be exchanged within each interaction",

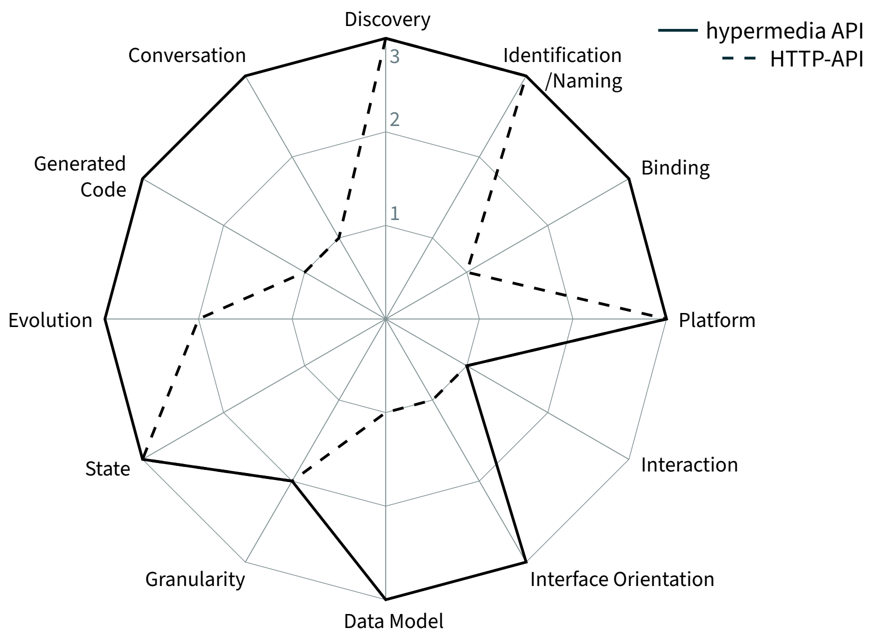

Figure 7.1: In contrast to a non-RESTful HTTP-API, the M&S hypermedia API supports loose coupling according to all but one facet. The numerical value 1 indicates *tight* coupling; 2 means *design-specific*, in other words dependent on specific implementations; and 3 indicates *loose* coupling.

and it is argued that fewer interactions through more coarse-grained interfaces result in more loosely coupled systems [76, p. 916]. The M&S hypermedia API offers both a relatively fine-granular interface through its choice of resources to be exposed and a coarse grained QPF interface for read-only access. It is thus classified as 'design-specific' since it depends on how a client interacts with the API. The interaction-facet is classified as synchronous and therefore 'tight': both the client and instance of the API need to be available at the same time for a successful interaction. This constraint is relaxed by several factors: first, the queue enables successful completion of requests even though no worker might be available temporarily. Second, the API's ability to decouple the successful completion of a request resulting in a time-consuming job from execution of said job, as for example when triggering a simulation by a `POST` request which immediately returns with a `201 Created` pointing to a resource which is being created in the background, can be seen as asynchronous. Third, Pautasso and Wilde [76, p. 915] suggest that caching also decreases coupling with respect to the interaction-facet through *non-blocking* (but still synchronous) interaction.

▲
▼ [94, section 6.1.2]

Improved Machine-actionability (H2.3; Q2)

The machine-actionability of the exposed M&S capabilities is evaluated qualitatively only: the PPA-implementation is able to use the M&S hypermedia API without being explicitly coded against it, as described in section 6.2.2. This successful use is seen as an indication of increased machine-actionability (H2.3). Moreover, since the FAIR principles explicitly include machine clients as their target audience, the increased FAIRness is also seen as a sign of increased machine-actionability. The fact that the PPA-implementation needs to ask a higher-level user for input that is compliant to certain shapes at run-time is *not* seen as an argument against machine-actionability for two reasons: first, the higher-level user could be software. Second, the user input has to be supplied eventually.

▴
▾ [94, section 6.1.1]

Improved FAIRness (H2; Q1)

The FAIRness of the M&S capabilities exposed as resources of a specific instance of the M&S hypermedia API are evaluated by comparison against the FAIRness of a .fmu-file and the REST-based HTTP-API exposing JSON representations that preceded the hypermedia API implementation.

For each of the 15 FAIR principles, the respective solution is classified as *not supported (numerical value 1)*, *supported (2)*, *partially implemented (3)* or *implemented (4)*. 'Not supported' means that a principle is not achievable due to conceptual discrepancies between the chosen architecture and the requirements of the principle. In contrast, 'supported' means that the principle is achievable, but not currently realized, either because it is not implemented; because it represents organizational issues (such as commitment to the long-term availability of metadata); or because it depends on user input. A score of 'partially implemented' means that either not all parts of a FAIR principle with multiple dimensions are realized; or that more could be done, for example by providing more detailed metadata, provenance information, or similar. Principles that are fully realized in the given implementation are classified as 'implemented'.

The categorization was done based on the FMI specification and the characteristics of the developed service. For example, principle I1 (formal language for knowledge representation) was categorized as 'not supported' for FMUs and 'implemented' for model representations of the hypermedia API: FMUs do not use a formal language for knowledge representation, whereas RDF is the data model recommended by the W3C for knowledge representation in the Semantic Web. As a second example, take principle R1 (richly described (meta)data): metadata could be added to FMUs as part of the `.modelDescription.xml`-file, but they are not required by the FMI standard and thus not always included in models that are exported as FMU. In the representations of models exposed in the hypermedia API, some metadata was added. Consequently, the values 'supported' and 'partially implemented' were assigned.

The results of the evaluation are visualized in figure 7.2. In the visualization, a higher value, plotted farther from the centre, corresponds to a higher degree of FAIRness. Figure 7.2 shows that there is no change in FAIRness for principles F2, F3 and R1.3 when comparing an FMU to the hypermedia API. F2 demands that "Data are described with rich metadata (defined by R1 below)", which is partially implemented in all variants: both an FMU and their representations as re-

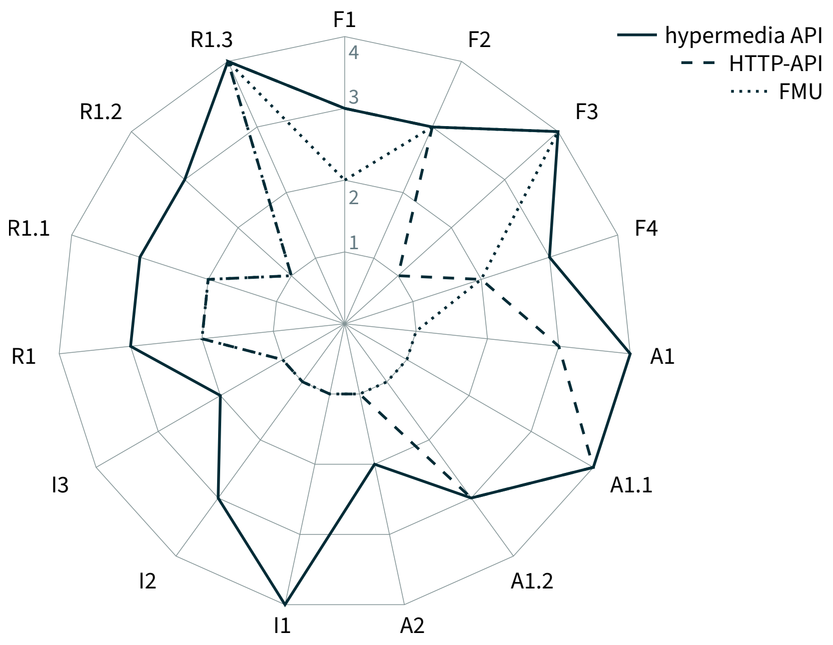

Figure 7.2: The FAIRness of M&S resources increases when exposed through a hypermedia API both compared to a .fmu-file and a non-RESTful HTTP-API.

sources contain some metadata; more “rich” metadata could be added through vendor annotations inside an FMU or by adding more triples to a hypermedia representation, but neither is currently implemented. R1.3 asks that “(Meta)data meet domain-relevant community standards”. FMI *is* the community standard for model exchange and co-simulation. Since all variants expose the metadata specified in the standard and allow downloading the original .fmu-file, R1.3 is classified as ‘implemented’.

The M&S hypermedia API fully implements principles F3 (explicit link between metadata and data), A1/A1.1 (open protocol), I1 (knowledge representation) and R1.3 (community standard). These are technical aspects that can be seen as the result of using the Semantic Web technology stack and basing the implementation on FMI.

Supported, but not even partially implemented, are principles A2 (long-lived metadata) and I3 (references to other (meta)data). Both depend on the application and context for which M&S capabilities are used and are potentially laborious; and neither is necessary for the applications presented in chapter 6.

All other aspects are partially implemented in the M&S hypermedia API, which shows the potential for reaching a high degree of FAIRness using the chosen approach. The classification is subjective and thus debatable; therefore details of the evaluation, including the notes on why a certain score was given, can be found in table A.2 and A.3 in appendix A.2. The principle I2, “(Meta)data use vocabularies that follow FAIR principles”, represents an exception: here, the FAIRness of the FMI- and SMS-ontologies is evaluated using the FOOPS! ontology pitfall scanner [36]. The other used ontologies were not evaluated because they cannot be influenced and because they are mostly well-known ontologies.

The REST-based HTTP-API also shows improved FAIRness compared to a .fmu-file due to the use of URLs to identify resources and making them available over HTTP (A1.x). However, FAIRness cannot be reached because no formal language for knowledge representation is used. It thus becomes hard to explicitly link metadata to the data it is about (F3 *not supported*); and to provide provenance and licensing information in a machine-readable way.

In conclusion, the FAIRness of M&S capabilities increases when exposed through a hypermedia API, both compared to not using an API and a REST-based HTTP-API. The FAIRness could be improved further, especially through organizational commitment and in-depth data modelling. Both are seen as beyond the scope of a proof of concept-implementation.

▲

Open-source PPA-implementation and -extension (H3.1)

Because it was needed to demonstrate the improved machine-actionability of the M&S capabilities and -entities exposed through the hypermedia API, the Pragmatic Proof Algorithm (section 5.4) by Verborgh et al. was implemented based on [119, 112, 117]. In these publications, using the PPA to generate thumbnails of images by posting them to a corresponding hypermedia API, served as an example to explain the algorithm. To verify that my implementation works correctly, this example was repeated.

Moreover, the example was modified such that the hypermedia API for thumbnail generation can expose its resources under different paths, essentially resulting in three different hypermedia APIs with the same functionality. Using the PPA-implementation *without* modification on these APIs demonstrates that the use of HATEOAS, in other words the generation of requests at run-time based on information obtained as part of the service interaction, makes clients more robust against changes in the service interface (as expected).

The source code was published on GitHub under the MIT licence; thus, others are granted the right to use the implementation for their own work. A README-document briefly explains usage as well as known issues. Consequently, H3.1 is seen as validated.

Proposed Extension of the PPA (H3.2)

Using the PPA as an agent to solve tasks involving models and their simulation necessitates that data conforming to requirements that depend on the models, and therefore only become known at run-time, are sent as part of requests. Therefore, a need for an extension of the PPA to account for this arises.

I devised and implemented an extension that depends on the existence of shape hints in the RESTdesc descriptions of the API (section 5.4.2). For example, the information "the body to be sent with this request is linked to a SHACL shapes graph (which is currently unknown)" is communicated in one RESTdesc rule, and the inform-

ation “the response to this request contains the definition of this shapes graph” in another. With this information, the PPA can create the necessary proofs to make sure that its goal is reachable *without* knowing what exactly is needed, and then ask a higher-level user for input at run-time after the shapes graph specifying the requirements was contained in the response to an earlier request. The extension is part of the published PPA-implementation and was first documented in [94, section 4.3.2].
Characteristics of the *extended* algorithm with regard to complexity and termination were not considered. Moreover, the implementation of asking a higher-level user for input must be improved as a workaround is currently used. For example, it should be possible to generate queries to a KG based on the shapes graphs.
Nevertheless, the extension presents one possible solution to an aspect that was not previously discussed. It works and H3.2 is therefore seen as validated.

Decreased Programming Effort (H3; Q4)

The programming effort decreases when asking the PPA to figure out requests and request sequence instead of manually coding them: When using the PPA to simulate a model instance, it is necessary to supply the URL of a M&S hypermedia API instance; the goal state; and the initial knowledge. Additionally, inputs matching the shapes must be provided when asked. In contrast, when using the REST-based HTTP-API, one first needs to instantiate a model; then read the OAS which has been re-generated to include the newly added model instance; and implement the request accordingly for every model used.
There are two main issues with the current state of the developed software. First, it is problematic that the `simulation`-resources need to be polled in order to learn about the existence of a simulation result, especially in combination with an implementation of the PPA as the service consumer: if the simulation takes longer to complete than it takes the PPA to request a representation of the `simulation`-resource, the resource representation will not contain the link to the result as expected due to the corresponding RESTdesc rule. Therefore, the execution of the request will not bring the PPA closer to reaching its goal. Consequently, the PPA will disregard the rule in future iterations and thus be unable to reach the goal even though it would have been possible had the request been sent *after* the simulation run completed. Second, the software was only tested to a limited extent.

▼ [94, section 6.2] ▲

Nonetheless, having an agent that can “operate” the M&S hypermedia API at run-time enables its use *at scale*, in other words with many different models that would each have required manual implementation before.

Value for MBSE Use Cases (H4; Q4)

To show that the concept of this work is both *feasible* and *useful*, a number of tasks from a MBSE context were solved using the developed software (chapter 6). Specifically, it was demonstrated that...

- a KG containing representations of FMUs in RDF allows finding models through *semantic queries*;
- the MSaaS-implementation facilitates *integration of simulation in distributed control processes*;
- the MSaaS-implementation enables the *parallel execution of simulations on a horizontally scalable set of computing resources* (ensemble forecast, component selection; section 6.2.1);
- the PPA-implementation works (thumbnail generation; section 6.2.2)
- the PPA-implementation in conjunction with the M&S hypermedia API allows *solving declaratively formulated MBSE tasks* (section 6.2.2); and
- a SE KG allows answering semantic queries about models *and their context* (section 6.3).

To conclude, the three issues that “unnecessarily complicate or even prevent” the reuse of models (section 4.1) were addressed as follows:

1. the FAIRness improved by providing them *as a service*, using the Semantic Web as execution context;
2. the interface heterogeneity as an obstacle to using the service at scale can be mitigated through the improved machine-actionability and loose coupling;
3. reducing the amount of inaccessible knowledge is facilitated and encouraged through the use of the RDF data model and an ecosystem of existing ontologies.

Moreover, the characteristics put forth as “desirable” were addressed as well. Data model and ontologies enable reasoning and knowledge queries and also facilitate encoding provenance data that is essential for traceability. Loose coupling and a declarative problem formulation are supported as a consequence of realizing the REST constraints. Last, the design of the service as a CNA and a methodological approach to DevOps represent the state of the art for developing SaaS.

7.2 Opportunities for Further Work

Despite the positive results discussed in-depth in the previous section, these results form but the basis for many interesting questions. Some directions for further research are outlined below. They are structured similarly to the hypotheses, focusing first on the individual building blocks (ontologies, parser; M&S hypermedia API; the PPA as a generic software agent) before turning to MBSE KGs and the wider MBSE context.

Ontologies to represent and reason about dynamic system models

Three perspectives on the further development of the suggested ontologies and the `fmi2rdf`-parser suggest themselves.

The first perspective is improving their functionality and robustness, as well as facilitating interoperability with higher-level ontologies. Measures to achieve this include...

- using a structured approach to ontology engineering, such as the Linked Open Terms (LOT) methodology [78];
- revising and extending both textual *definitions* and the *axiomatization* of classes;
- explicitly stating the allowed OWL constructs, in other words deciding on an OWL profile and its associated complexity;
- improving test coverage of `fmi2rdf` and upgrading to version 3.x of the FMI standard;
- revising the handling of units and types, building upon existing approaches; and
- facilitating interoperability by aligning the ontologies with mid-level engineering ontologies (and, by extension, a top-level ontology) serving as a proven mental framework, such as the Industrial Ontologies Foundry (IOF) core ontology [54].

Second, adoption and reuse of the ontologies should be inspired through demonstrating their value by using them. For example, aspects that ought to be addressed are...

- building a collection of models that explicitly provide information about their purpose and conditions for which they are valid;
- executing semantic queries on such a collection of models;
- using reasoning to handle automatic alignment and conversion of types and units; and
- investigating questions of consistency and composability for FMUs by means of reasoning, as for example discussed in detail by Tudorache [110].

Third, it would be interesting to explore ontology-driven modelling on the basis of the FMI- and SMS-ontologies by means of UCs. Most likely, these would raise the issue of reasoning under SNAF as well as the need for methodological ontologies for related standards such as SSP [67].

Hypermedia APIs to expose/access M&S entities and -capabilities

For the M&S hypermedia API, realizing more of its potential through applications and moving towards a more dependable and more secure implementation are logical next steps.

For direct application, the following scenarios are suitable:

- use as a building block in a SOA in which M&S capabilities are required;
- exploiting parallelization and dynamic, horizontal scaling for parameter fitting and optimization problems;
- managing and developing a collection of models centred around a topic, for example by research groups to facilitate the continuity of their work; and

  – use as a backend to realize human-facing web applications, such as supporting house owners with investment decisions regarding the supply of electric and thermal energy using renewable sources, *tailored* to their specific prerequisites and needs.

Regarding the robustness of the service implementation, an analysis of threats to dependability and security should reveal the next steps to moving from a proof-of-concept to production-ready software. Regarding features, lifting restrictions on FMUs (section 5.3.2); implementing the “measured service” characteristics (likely using sidecar containers) in conjunction with usage policies and access control; as well as implementing a provenance trail throughout the application should provide additional value for both service provider and -consumer.

▾ [94, section 7]

In this work, dynamic system models and their simulation were focused. However, it should be possible to extend the ideas of this work and the service concept to other types of models. The prerequisite for direct integration into the developed hypermedia API is that it must be possible to meaningfully represent these models and their simulation through the conceptual resources chosen as the service interface. If this is the case, implementation should be straightforward because of the distinction between the format-independent API component and the format-dependent worker implementations: for new model formats, new worker implementations, including the creation of a representation of the model format in RDF, are required—but no changes to the API component.

▴

PPA and RESTdesc to realize generic software agents

As shown in [119, 2], N3 rules and proof-based algorithms such as the PPA show potential for solving problems in complex, frequently changing environments. Consequently, it would be interesting to devise and implement UCs that benefit from the approach and combine hypermedia APIs, sensors, and actuators with generic software agents.

Moreover, these environments would be well suited to put into practice the feature-based approach to API development [114] and, consequently, the creation of agents that use them.

Knowledge graphs in MBSE

Last, besides the straightforward uses of KGs, an MBSE KG suggests itself for finding parameters for model instances or input data sets for simulations, possibly based on the shapes communicated by the M&S hypermedia API at run-time. Moreover, it could help with making MBSE processes traceable.

8. Conclusion

The work presented in this thesis explores the use of Semantic Web concepts and -technologies to provide M&S entities and -capabilities *as a service*. The approach is seen as one possible way to address two core problems that hinder a more widespread reuse of models and their simulation: a lack of FAIRness and the heterogeneity of model interfaces (which makes the reuse at scale expensive through the programming effort necessary). This chapter concludes the research project by summarizing from a high-level perspective its hypotheses and achievements; the opportunities for further work; and its contributions.

First and foremost, it was hypothesized that a cloud-native implementation of the MSaaS-concept based on FMI in the form of a hypermedia API would lead to improved FAIRness and facilitate the use of M&S in distributed settings *at scale* through improved machine-actionability. From a conceptual point of view, the design of the service interface would allow the service's use in loosely coupled systems. The representation of FMUs in RDF would enable answering semantic queries on them, which was not possible using open-source components before. Furthermore, UCs would point out the potential value of the approach for MBSE.

All hypotheses could be validated by implementing the necessary ontologies and software and using them to realize the UCs. Researchers and software engineers are enabled to review and reuse the developed source code because it is released publicly under permissive open-source licences.

Four directions for further work suggest themselves:

- **demonstrating the value of the created building blocks as they are**, with the intent of **inspiring reuse and adoption by others** (building a model collection; using the service in a SOA (exploiting parallelization/horizontal scaling; as backend); using the PPA-implementation; building useful KGs; ...);
- **analyzing threats to dependability and security of the software**, as a first step towards production-readiness;
- **adding and refining functionality** (axiomatization and reasoning; representation/conversion of units; alignment with mid-level ontologies; service metrics; provenance data; ...); and
- **further research** (determining composability of models through semantics and reasoning; ontology-driven modelling; using self-descriptive hypermedia APIs and -sensors in changing environments in conjunction with generic software agents, such as the PPA, as well as knowledge graphs; ...).

Because of the positive results and promising directions to further develop the underlying ideas based on these results, the approach can be seen as both interesting and relevant for practice and research. Overarching topics that the work is related to—and could contribute to—include distributed MBSE processes; breaking up knowledge silos and working together interdisciplinarily; realizing the potential of M&S; and the FAIRness of digital assets in general.

However, technical and organizational challenges are expected when attempting to implement the approach of this thesis outside academia. Technical challenges that are likely to occur are bringing together data from various sources in a coherent, sustainable way (raising the question of which ontologies to use); and making all software dependable and secure. Given the necessary expertise, these challenges are solvable. In contrast, it could be more difficult to overcome organizational hurdles, which include convincing non-technical stakeholders to commit to the approach by explaining its benefits in an understandable way; establishing processes to make implicit knowledge that typically only exists in the minds of developers/engineers, explicit; and facilitating the change in mindset required from programmers (from fixed contracts to a graph data model and queries, [112, p. 97]).

To conclude, this thesis showed that the chosen approach is indeed logical and promising; novel; feasible; and directly useful. It is logical and promising because the FAIR principles are widely accepted as desirable, and Semantic Web technologies suggest themselves for improving the FAIRness of digital assets. Moreover, CNAs represent the state of technology for realizing services in the Web, and their defining characteristics are now expected by users. In my eyes, dynamic system models and their simulation should not be excluded from these developments. It is new because, to the best of my knowledge, the chosen combination of concepts and technologies has not been investigated in detail before. The UCs show feasibility and usefulness.

Personally, I believe that the approach should be applied to solve real-world problems for three main reasons. First, I think that self-descriptive hypermedia APIs and generic software agents provide a level of abstraction that is suitable to deal with complex, frequently changing systems of interconnected systems that involve sensors, actuators, models, and simulations. Moreover, SOA and REST account for the fact that reality is an open world distributed among many stakeholders, which makes enforcing centralized interface designs impossible and aiming for loose coupling desirable. Through the use of a distributed data model and ontologies, the need for integrating data and processes that are based on different perspectives and possibly incompatible assumptions can be met. Consequently, I suspect that the approach investigated in this work can help with realizing MBSE processes that are traceable, explainable, secure, and dependable.

Perhaps others also see value in the ideas underlying this thesis, and build upon my work with their own unique background and perspective.

References

[1] Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Modeling in RDF, RDFS and OWL*. Morgan Kaufmann Publishers, 2009. ISBN: 978-0-12-373556-0.

[2] Dörthe Arndt et al. “SENSdesc: Connect Sensor Queries and Context”. In: *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies - AI4Health*, INSTICC. SciTePress, 2018, pp. 671–679. DOI: 10.5220/0006733106710679.

[3] Donovan Artz and Yolanda Gil. “A survey of trust in computer science and the Semantic Web”. In: *Journal of Web Semantics* 5.2 (2007). Software Engineering and the Semantic Web, pp. 58–71. ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.03.002. URL: http://www.sciencedirect.com/science/article/pii/S1570826807000133.

[4] Algirdas Avizienis et al. “Basic Concepts and Taxonomy of Dependable and Secure Computing”. In: *IEEE Trans. Dependable Secur. Comput.* 1.1 (Jan. 2004), pp. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2.

[5] Jakob Axelsson. “Achieving System-of-Systems Interoperability Levels Using Linked Data and Ontologies”. In: *INCOSE International Symposium* 30.1 (2020), pp. 651–665. DOI: 10.1002/j.2334-5837.2020.00746.x.

[6] David Beckett. *RDF 1.1 N-Triples. A line-based syntax for an RDF graph*. W3C Recommendation. W3C, 25th Feb. 2014. URL: http://www.w3.org/TR/2014/REC-n-triples-20140225/.

[7] David Beckett et al. *RDF 1.1 Turtle. Terse RDF Triple Language*. W3C Recommendation. W3C, 25th Feb. 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/.

[8] David Bell et al. “Service-oriented simulation using web ontology”. In: *International Journal of Simulation and Process Modelling* 7.3 (2012), pp. 217–227. DOI: 10.1504/IJSPM.2012.049148.

[9] Tim Berners-Lee, Roy T. Fielding and Larry M. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. Request for Comments (RFC) 3986. Jan. 2005. DOI: 10.17487/RFC3986. URL: https://www.rfc-editor.org/info/rfc3986.

[10] Tim Berners-Lee, James Hendler and Ora Lassila. “The Semantic Web”. In: *Scientific American* 284 (May 2001), pp. 35–43. DOI: 10.1038/scientificamerican0501-34. URL: https://lassila.org/publications/2001/SciAm.html.

[11] Abraham Bernstein and Natasha Noy. *Is This Really Science? The Semantic Webber’s Guide to Evaluating Research Contributions*. Tech. rep. IFI-2014.02. 2014, pp. 1–18. URL: https://www.merlin.uzh.ch/contributionDocument/download/6915 (visited on 03/08/2023).

[12] Stefan Bittner, Olaf Oelsner and Thomas Neidhold. “Using FMI in a cloud-based Web Application for System Simulation”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press, 18th Sept. 2015. DOI: 10.3384/ecp15118845.

[13] Chris Bizer and Richard Cyganiak. *RDF 1.1 TriG. RDF Dataset Language*. W3C Recommendation. W3C, 25th Feb. 2014. URL: https://www.w3.org/TR/2014/REC-trig-20140225/.

[14] *RDF Schema 1.1*. W3C Recommendation. W3C, 25th Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[15] Carlos Buil-Aranda et al. “SPARQL Web-Querying Infrastructure: Ready for Action?” In: *The Semantic Web – ISWC 2013*. Ed. by Harith Alani et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 277–293.

[16] Rafael C. Cardoso and Angelo Ferrando. “A Review of Agent-Based Programming for Multi-Agent Systems”. In: *Computers* 10.2 (2021). ISSN: 2073-431X. DOI: 10.3390/computers10020016. URL: https://www.mdpi.com/2073-431X/10/2/16.

[17] *RDF 1.1 N-Quads. A line-based syntax for RDF datasets*. W3C Recommendation. W3C, 25th Feb. 2014. URL: http://www.w3.org/TR/2014/REC-n-quads-20140225/.

[18] Erdal Cayirci. “A joint trust and risk model for MSaaS mashups”. In: *2013 Winter Simulations Conference (WSC)*. IEEE, Dec. 2013, pp. 1347–1358. DOI: 10.1109/WSC.2013.6721521.

[19] Erdal Cayirci. “Modeling and simulation as a cloud service: A survey”. In: *2013 Winter Simulations Conference (WSC)*. IEEE, Dec. 2013, pp. 389–400. DOI: 10.1109/wsc.2013.6721436.

[20] François E. Cellier. *Continuous System Modeling*. Springer Science+Business Media Inc., 1991. ISBN: 978-0-387-97502-3. DOI: 10.1007/978-1-4757-3922-0.

[21] François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer Science+Business Media Inc., 2006. ISBN: 978-0-387-26102-7. DOI: 10.1007/0-387-30260-3.

[22] Robert M. Cubert and Paul A. Fishwick. “A Framework for Distributed Object-oriented Multimodeling and Simulation”. In: *Proceedings of the 29th Conference on Winter Simulation*. WSC ’97. Atlanta, Georgia, USA: IEEE Computer Society, 1997, pp. 1315–1322. DOI: 10.1145/268437.268777.

[23] *Empirical*. In: *Dictionary.com online dictionary*. Ed. by Dictionary.com. URL: https://www.dictionary.com/browse/empirical (visited on 29/06/2023).

[24] *Metadata*. In: *Dictionary.com online dictionary*. Ed. by Dictionary.com. URL: https://www.dictionary.com/browse/metadata (visited on 29/06/2023).

[25] *Predicate*. In: *Dictionary.com online dictionary*. Ed. by Dictionary.com. URL: https://www.dictionary.com/browse/predicate (visited on 25/10/2023).

[26] Anastasia Dimou et al. “RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data”. In: *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014*. Ed. by Christian Bizer et al. Vol. 1184. CEUR Workshop Proceedings. CEUR-WS.org, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014%5C_paper%5C_01.pdf.

[27] EDN, ed. *Genetic algorithm solves thermistor-network component values*. 19th Mar. 2008. URL: https://www.edn.com/genetic-algorithm-solves-thermistor-network-component-values (visited on 08/05/2021).

[28] Hilding Elmqvist, Martin Malmheden and Johan Andreasson. “A Web Architecture for Modeling and Simulation”. In: *Proceedings of the 2nd Japanese Modelica Conference Tokyo, Japan, May 17-18, 2018*. Linköping University Electronic Press, 21st Feb. 2019. DOI: 10.3384/ecp18148255.

[29] Leonardo Ferreira Leite et al. “A Survey of DevOps Concepts and Challenges”. In: *ACM Computing Surveys* 52 (Nov. 2019), pp. 1–35. DOI: 10.1145/3359981.

[30] Roy T. Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. PhD thesis. Irvine, CA, USA: University of California, Irvine, 2000. URL: https://roy.gbiv.com/pubs/dissertation/top.htm.

[31] Roy T. Fielding. *REST APIs must be hypertext-driven*. 20th Oct. 2008. URL: https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (visited on 11/01/2022).

[32] Roy T. Fielding, Mark Nottingham and Julian Reschke. *HTTP Semantics*. Request for Comments (RFC) 9110. June 2022. DOI: 10.17487/RFC9110. URL: https://www.rfc-editor.org/info/rfc9110.

[33] Paul A. Fishwick. “Web-based Simulation”. In: *Proceedings of the 29th Conference on Winter Simulation*. WSC ’97. Atlanta, Georgia, USA: IEEE Computer Society, 1997, pp. 100–102. DOI: 10.1145/268437.268457.

[34] Félix-Antoine Fortin et al. “DEAP: Evolutionary Algorithms Made Easy”. In: *Journal of Machine Learning Research* 13.70 (July 2012), pp. 2171–2175. URL: http://jmlr.org/papers/v13/fortin12a.html.

[35] Martin Fowler and James Lewis. *Microservices. A definition of this new architectural term*. 25th Mar. 2014. URL: https://martinfowler.com/articles/microservices.html (visited on 01/11/2023).

[36] Daniel Garijo, Oscar Corcho and Marıa Poveda-Villalón. “FOOPS!: An Ontology Pitfall Scanner for the FAIR Principles”. In: CEUR Workshop Proceedings 2980 (2021). URL: http://ceur-ws.org/Vol-2980/paper321.pdf.

[37] Birte Glimm and Heiner Stuckenschmidt. “15 Years of Semantic Web: An Incomplete Survey”. In: *KI – Künstliche Intelligenz* 30.2 (June 2016), pp. 117–130. ISSN: 1610-1987. DOI: 10.1007/s13218-016-0424-1.

[38] Susanne Hall, Cary Moskovitz and Michael Pemberton. *Text Recycling. TRRP Best Practices for Researchers*. Apr. 2021. eprint: https://textrecycling.org/files/2021/04/TRRP_Best-Practices-for-Researchers.pdf. URL: https://textrecycling.org/resources/best-practices-for-researchers/.

[39] Jo Erskine Hannay and Tom van den Berg. “The NATO MSG-136 Reference Architecture for M&S as a Service”. In: *Proceedings of the NATO modelling and simulation group symposium on M&S technologies and standards for enabling alliance interoperability and pervasive M&S Applications* (Lisbon, Portugal, 19th–20th Oct. 2017). STO-MP-MSG-149. Oct. 2017, p. 3.

[40] Jo Erskine Hannay et al. “Modeling and Simulation as a Service infrastructure capabilities for discovery, composition and execution of simulation services”. In: *The Journal of Defense Modeling and Simulation. Applications, Methodology, Technology* (2020). DOI: 10.1177/1548512919896855.

[41] *SPARQL 1.1 Query Language*. W3C Recommendation. W3C, 21st Mar. 2013. URL: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[42] Pascal Hitzler. “A Review of the Semantic Web Field”. In: *Communications of the ACM* 64.2 (Feb. 2021), pp. 76–83. ISSN: 0001-0782. DOI: 10.1145/3397512.

[43] Patrick Hochstenbach, Jos De Roo and Ruben Verborgh. “RDF Surfaces: Computer Says No”. In: *Proceedings of the 1st Workshop on Trusting Decentralised Knowledge Graphs and Web Data*. May 2023. arXiv: 2305.08476.

[44] M. Hofmann, J. Palii and G. Mihelcic. “Epistemic and normative aspects of ontologies in modelling and simulation”. In: *Journal of Simulation* 5.3 (Aug. 2011), pp. 135–146. DOI: 10.1057/jos.2011.13.

[45] Aidan Hogan et al. “Knowledge Graphs”. In: *ACM Computing Surveys* 54.4 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3447772.

[46] Ian Jacobs and Norman Walsh. *Architecture of the World Wide Web, Volume One*. W3C Recommendation. W3C, 15th Dec. 2004. URL: https://www.w3.org/TR/2004/REC-webarch-20041215/.

[47] Jad El-khoury. *An Analysis of the OASIS OSLC Integration Standard, for a Cross-disciplinary Integrated Development Environment: Analysis of market penetration, performance and prospects*. Tech. rep. KTH, Mechatronics, 2020, p. 55. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-272834.

[48] Sabrina Kirrane. "Intelligent software web agents: A gap analysis". In: *Journal of Web Semantics* 71 (2021), p. 100659. ISSN: 1570-8268. DOI: 10.1016/j.websem.2021.100659. URL: https://www.sciencedirect.com/science/article/pii/S1570826821000342.

[49] Sabrina Kirrane and Stefan Decker. "Intelligent Agents: The Vision Revisited". In: *Proceedings of the 2nd Workshop on Decentralizing the Semantic Web co-located with the 17th International Semantic Web Conference, DeSemWeb@ISWC 2018, Monterey, California, USA, October 8, 2018*. Ed. by Ruben Verborgh, Tobias Kuhn and Tim Berners-Lee. Vol. 2165. CEUR Workshop Proceedings. CEUR-WS.org, 2018. URL: http://ceur-ws.org/Vol-2165/paper2.pdf.

[50] Christian König et al. "Traceability in the Model-based Design of Cyber-Physical Systems". In: *Proceedings of the American Modelica Conference 2020, Boulder, Colorado, USA, March 23-25, 2020*. Linköping University Electronic Press, Nov. 2020. DOI: 10.3384/ecp20169168.

[51] Dimitris Kontokostas and Holger Knublauch. *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C, July 2017. URL: https://www.w3.org/TR/2017/REC-shacl-20170720/.

[52] Nane Kratzke and Peter-Christian Quint. "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study". In: *Journal of Systems and Software* 126 (2017), pp. 1–16. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.01.001. URL: http://www.sciencedirect.com/science/article/pii/S0164121217300018.

[53] Nane Kratzke and Robert Siegfried. "Towards cloud-native simulations – lessons learned from the front-line of cloud computing". In: *The Journal of Defense Modeling and Simulation. Applications, Methodology, Technology* 18 (1 2021), pp. 39–58. DOI: 10.1177/1548512919895327. URL: https://journals.sagepub.com/doi/10.1177/1548512919895327.

[54] Boonserm Kulvatunyou et al. "The Industrial Ontologies Foundry (IOF) Core Ontology". en. In: Formal Ontologies Meet Industry (FOMI) 2022. Tarbes, France, Sept. 2022. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=935068.

[55] Claude Lacoursière and Tomas Härdin. "FMI Go! A simulation runtime environment with a client server architecture over multiple protocols". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping University Electronic Press, 4th July 2017. DOI: 10.3384/ecp17132653.

[56] Markus Lanthaler. *Hydra Core Vocabulary. A Vocabulary for Hypermedia-Driven Web APIs*. Tech. rep. Version Unofficial Draft 13 July 2021. July 2021. URL: http://www.hydra-cg.com/spec/latest/core/.

[57] Markus Lanthaler, David Wood and Richard Cyganiak. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C, 25th Feb. 2014. URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[58] D.W. McKee et al. "The Internet of Simulation, a Specialisation of the Internet of Things with Simulation and Workflow as a Service (SIM/WFaaS)". In: *11th IEEE International Symposium on Service-Oriented System Engineering (SOSE 2017)*. IEEE, Jan. 2017. URL: http://eprints.whiterose.ac.uk/111418/.

[59] *Measuring digital development: Facts and Figures 2022*. Report. International Telecommunication Union (ITU), 30th Nov. 2022. URL: https://www.itu.int/itu-d/reports/statistics/facts-figures-2022/.

[60] Ben De Meester et al. “Detailed Provenance Capture of Data Processing”. In: *Proceedings of the First Workshop on Enabling Open Semantic Science (SemSci)*. 2017, pp. 31–38. URL: https://ceur-ws.org/Vol-1931/paper-05.pdf.

[61] Christoph Meinel. *‚Selbstplagiat' und gute wissenschaftliche Praxis*. Aug. 2013. DOI: 10.13140/RG.2.2.30238.46406.

[62] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145. National Institute of Standards and Technology, 2011. DOI: 10.6028/NIST.SP.800-145. URL: https://www.nist.gov/publications/nist-definition-cloud-computing.

[63] *Serendipity*. In: *Merriam-Webster.com Dictionary*. Ed. by Merriam-Webster. URL: https://www.merriam-webster.com/dictionary/serendipity (visited on 18/07/2023).

[64] Matthias Mitterhofer et al. “An FMI-enabled methodology for modular building performance simulation based on Semantic Web Technologies”. In: *Building and Environment* 125 (2017), pp. 49–59. ISSN: 0360-1323. DOI: 10.1016/j.buildenv.2017.08.021. URL: http://www.sciencedirect.com/science/article/pii/S0360132317303736.

[65] Modelica Association. *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0.2*. Tech. rep. Linköping: Modelica Association, Dec. 2020. URL: https://fmi-standard.org.

[66] Modelica Association. *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association, Feb. 2021. URL: https://specification.modelica.org/maint/3.5/MLS.html.

[67] Modelica Association. *System Structure and Parameterization Version 1.0*. Tech. rep. Version 1.0. Modelica Association, Mar. 2019. URL: https://ssp-standard.org/.

[68] William Morgan. *The Service Mesh: What Every Software Engineer Needs to Know about the World's Most Over-Hyped Technology*. 9th Nov. 2019. URL: https://buoyant.io/service-mesh-manifesto (visited on 07/05/2021).

[69] *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. W3C, 11th Dec. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[70] Specialist Team MSG-131. *Modelling and Simulation as a Service: New Concepts and Service-Oriented Architectures*. Final Report AC/323(MSG-131)TP/608. North Atlantic Treaty Organization NATO, 2015. URL: https://www.sto.nato.int/publications/STO%20Technical%20Reports/STO-TR-MSG-131/$$TR-MSG-131-ALL.pdf.

[71] Mark A. Musen. “The Protégé Project: A Look Back and a Look Forward”. In: *AI Matters* 1.4 (June 2015), pp. 4–12. ISSN: 2372-3483. DOI: 10.1145/2757001.2757003.

[72] OASIS. *OSLC Core Version 3.0. Part 1: Overview*. OASIS Standard. OASIS, 26th Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/oslc-core.html.

[73] OASIS. *OSLC Core Version 3.0. Part 3: Resource Preview*. OASIS Standard. OASIS, 26th Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/resource-preview.html.

[74] OASIS. *OSLC Core Version 3.0. Part 4: Delegated Dialogs*. OASIS Standard. OASIS, 26th Aug. 2021. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/dialogs.html.

[75] OASIS. *Reference Model for Service Oriented Architecture 1.0*. OASIS Standard. OASIS Open, 12th Oct. 2006. URL: http://docs.oasis-open.org/soa-rm/v1.0/.

[76] Cesare Pautasso and Erik Wilde. “Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design”. In: *18th International World Wide Web Conference*. Apr. 2009, pp. 911–920. URL: http://www2009.eprints.org/92/.

[77] Martin Pelikan. “Genetic Algorithms”. In: *Wiley Encyclopedia of Operations Research and Management Science*. American Cancer Society, 2011. DOI: 10.1002/9780470400531.eorms0357.

[78] María Poveda-Villalón et al. “LOT: An industrial oriented ontology engineering framework”. In: *Engineering Applications of Artificial Intelligence* 111 (2022), p. 104755. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2022.104755. URL: https://www.sciencedirect.com/science/article/pii/S0952197622000525.

[79] Judicaël Ribault and Gabriel Wainer. “Using Workflows and Web Services to Manage Simulation Studies”. In: *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*. TMS/DEVS ’12. Orlando, Florida: Society for Computer Simulation International, 2012, 50:1–50:6. URL: http://dl.acm.org/citation.cfm?id=2346616.2346666.

[80] Carlos Rodríguez et al. “REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices”. In: *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*. Ed. by Alessandro Bozzon, Philippe Cudre-Maroux and Cesare Pautasso. Cham: Springer International Publishing, 2016, pp. 21–39. DOI: 10.1007/978-3-319-38791-8_2.

[81] Jos De Roo and Patrick Hochstenbach. *Notation3 Language*. W3C Community Group Draft Report. W3C, 23rd Oct. 2023. URL: https://w3c.github.io/N3/spec/.

[82] Jennifer Rowley. “The wisdom hierarchy: representations of the DIKW hierarchy”. In: *Journal of Information Science* 33.2 (2007), pp. 163–180. DOI: 10.1177/0165551506070706.

[83] Sebastian Rudolph. “Foundations of Description Logics”. In: *Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*. Ed. by Axel Polleres et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 76–136. ISBN: 978-3-642-23032-5. DOI: 10.1007/978-3-642-23032-5_2.

[84] Thomas Schmitt and Markus Andres. *Methoden zur Modellbildung und Simulation mechatronischer Systeme. Bondgraphen, objektorientierte Modellierungstechniken und numerische Integrationsverfahren*. Springer Vieweg, 2019. ISBN: 978-3-658-25088-1. DOI: 10.1007/978-3-658-25089-8.

[85] Thomas Schmitt et al. “A Novel Proposal on how to Parameterize Models in Dymola Utilizing External Files under Consideration of a Subsequent Model Export using the Functional Mock-Up Interface”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press, 18th Sept. 2015. DOI: 10.3384/ecp1511823. URL: https://2015.international.conference.modelica.org/proceedings/html/errata/errata_SchmittAndresZieglerDiehl.pdf. Revised version.

[86] Michael Schneider. *OWL 2 Web Ontology Language. RDF-Based Semantics (Second Edition)*. W3C Recommendation. W3C, 11th Dec. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/.

[87] *RDF 1.1 Primer*. W3C Working Group Note. W3C, 24th June 2014. URL: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[88] Mojtaba Shahin, M. Ali Babar and Muhammad Aufeef Chauhan. “Architectural Design Space for Modelling and Simulation as a Service: A Review”. In: *Journal of Systems and Software* 170 (24th July 2020), p. 110752. ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110752. URL: http://www.sciencedirect.com/science/article/pii/S0164121220301746.

[89] Shashank Shekhar et al. “A simulation as a service cloud middleware”. In: *Annals of Telecommunications* 71.3 (Apr. 2016), pp. 93–108. ISSN: 1958-9395. DOI: 10.1007/s12243-015-0475-6.

[90] Gregory A. Silver et al. “DeMO: An Ontology for Discrete-event Modeling and Simulation”. In: *SIMULATION* 87.9 (2011). PMID: 22919114, pp. 747–773. DOI: 10.1177/0037549710386843.

[91] *Sorbonne declaration on research data rights*. Jan. 2020. URL: https://www.leru.org/files/Sorbonne-declaration.pdf.

[92] Moritz Stüber, Lukas Exel and Georg Frey. “Using Modelling and Simulation as a Service (MSaaS) for Facilitating Flexibility-based Optimal Operation of Distribution Grids”. In: *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO*. INSTICC. SciTePress, 2018, pp. 613–620. DOI: 10.5220/0006899106230630.

[93] Moritz Stüber and Georg Frey. “A Cloud-native Implementation of the Simulation as a Service-Concept Based on FMI”. In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping University Electronic Press, Sept. 2021. DOI: 10.3384/ecp21181393.

[94] Moritz Stüber and Georg Frey. “Dynamic system models and their simulation in the Semantic Web”. In: *Semantic Web Journal* (June 2023). Ed. by Bahar Aameri et al., pp. 1–36. DOI: 10.3233/sw-233359.

[95] Moritz Stüber and Georg Frey. “FAIRness für Modelle und Simulationen”. In: *atp magazin* 63.10 (Oct. 2022), pp. 58–65. DOI: 10.17560/atp.v63i10.2626.

[96] Moritz Stüber and Georg Frey. “FAIRness in der Automatisierungstechnik am Beispiel Modellierung und Simulation”. In: *Automation 2022*. VDI Verlag, 2022, pp. 343–354. DOI: 10.51202/9783181023990-343.

[97] Moritz Stüber et al. “Forecast Quality of Physics-Based and Data-Driven PV Performance Models for a Small-Scale PV System”. In: *Frontiers in Energy Research* 9 (2021), p. 108. ISSN: 2296-598X. DOI: 10.3389/fenrg.2021.639346. URL: https://www.frontiersin.org/article/10.3389/fenrg.2021.639346.

[98] *Systems Engineering Vision 2020*. Tech. rep. INCOSE-TP-2004-004-02. Version 2.03. International Council on Systems Engineering INCOSE, Sept. 2007. URL: https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf.

[99] Ruben Taelman. *Quad Pattern Fragments. A low-cost, queryable Linked Data Fragments interface supporting quads*. Unofficial Draft. 14th Sept. 2020. URL: https://linkeddatafragments.org/specification/quad-pattern-fragments/.

[100] Ruben Taelman et al. “Comunica: a Modular SPARQL Query Engine for the Web”. In: *Proceedings of the 17th International Semantic Web Conference*. Oct. 2018. URL: https://comunica.github.io/Article-ISWC2018-Resource/.

[101] Simon J. E. Taylor et al. “Grand challenges for modeling and simulation: simulation everywhere—from cyberinfrastructure to clouds to citizens”. In: *SIMULATION* 91.7 (2015), pp. 648–665. DOI: 10.1177/0037549715590594.

[102] Michael Tiller. “Vehicle Thermal Management – A Case Study in Web-Based Engineering Analysis”. In: *Proceedings of the 10th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 96. Linköping University Electronic Press, 2014, pp. 1073–1079. DOI: 10.3384/ecp140961073.

[103] Michael Tiller and Dietmar Winkler. “modelica.university: A Platform for Interactive Modelica Content”. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping University Electronic Press, 4th July 2017, pp. 725–734. DOI: 10.3384/ecp17132725.

[104] Michael Tiller and Dietmar Winkler. “Where impact got Going”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press, 18th Sept. 2015. DOI: 10.3384/ecp15118725.

[105] Alexander Toet, Susanne Tak and Jan Erp. “Visualizing uncertainty. Towards a better understanding of weather forecasts”. In: *Tijdschrift voor Human Factors* 41 (Apr. 2016), pp. 9–14.

[106] Andreas Tolk. “Interoperability, Composability, and Their Implications for Distributed Simulation: Towards Mathematical Foundations of Simulation Interoperability”. In: *Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. DS-RT ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 3–9. DOI: 10.1109/DS-RT.2013.8.

[107] Andreas Tolk. “The Elusiveness of Simulation Interoperability—What is Different From Other Interoperability Domains?” In: *2018 Winter Simulation Conference (WSC)*. Gothenburg, Sweden, Dec. 2018, pp. 679–690. DOI: 10.1109/WSC.2018.8632363.

[108] Andreas Tolk and John A. Miller. “Enhancing simulation composability and interoperability using conceptual/semantic/ontological models”. In: *Journal of Simulation* 5.3 (Aug. 2011), pp. 133–134. DOI: 10.1057/jos.2011.18.

[109] Andreas Tolk and Saurabh Mittal. “A Necessary Paradigm Change to Enable Composable Cloud-Based M&S Services”. In: *Proceedings of the 2014 Winter Simulation Conference*. WSC ’14. Savannah, Georgia: IEEE Press, 2014, pp. 356–366.

[110] Tania Tudorache. “Employing Ontologies for an Improved Development Process in Collaborative Engineering”. PhD thesis. Berlin, Germany: Technische Universität Berlin, Nov. 2006. DOI: 10.14279/depositonce-1477.

[111] Ruben Verborgh. “Piecing the puzzle — Self-publishing queryable research data on the Web”. In: *Proceedings of the 10th Workshop on Linked Data on the Web*. Ed. by Sören Auer et al. Vol. 1809. CEUR Workshop Proceedings. Apr. 2017. URL: https://ruben.verborgh.org/articles/queryable-research-data/.

[112] Ruben Verborgh. “Serendipitous Web Applications through Semantic Hypermedia”. PhD thesis. Ghent, Belgium: Ghent University, Feb. 2014. URL: https://ruben.verborgh.org/phd/ruben-verborgh-phd.pdf.

[113] Ruben Verborgh. *Turtles all the way down. APIs are more than just data: context and controls also belong in the message*. 6th Oct. 2015. URL: https://ruben.verborgh.org/blog/2015/10/06/turtles-all-the-way-down/ (visited on 12/01/2022).

[114] Ruben Verborgh and Michel Dumontier. “A Web API ecosystem through feature-based reuse”. In: *Internet Computing* 22.3 (May 2018), pp. 29–37. DOI: 10.1109/MIC.2018.032501515. URL: https://ruben.verborgh.org/articles/web-api-ecosystem/.

[115] Ruben Verborgh and Jos De Roo. “Drawing Conclusions from Linked Data on the Web: The EYE Reasoner”. In: *IEEE Softw.* 32.3 (2015), pp. 23–27. DOI: 10.1109/MS.2015.63.

[116] Ruben Verborgh and Miel Vander Sande. “The Semantic Web identity crisis: in search of the trivialities that never were”. In: *Semantic Web Journal* 11.1 (Jan. 2020), pp. 19–27. DOI: 10.3233/SW-190372. URL: https://ruben.verborgh.org/articles/the-semantic-web-identity-crisis/.

[117] Ruben Verborgh et al. “Linked Data and Linked APIs: Similarities, Differences, and Challenges”. In: *The Semantic Web: ESWC 2012 Satellite Events*. Ed. by Elena Simperl et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 272–284.

[118] Ruben Verborgh et al. “The fallacy of the multi-API culture: Conceptual and practical benefits of Representational State Transfer (REST)”. In: *Journal of Documentation* 71.2 (2015), pp. 233–252. DOI: 10.1108/JD-07-2013-0098.

[119] Ruben Verborgh et al. “The Pragmatic Proof: Hypermedia API Composition and Execution”. In: *Theory and Practice of Logic Programming* 17.1 (2017), pp. 1–48. DOI: 10.1017/S1471068416000016. URL: http://arxiv.org/pdf/1512.07780v1.pdf.

[120] Ruben Verborgh et al. “Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web”. In: *Journal of Web Semantics* 37–38 (Mar. 2016), pp. 184–206. ISSN: 1570-8268. DOI: 10.1016/j.websem.2016.03.003. URL: http://linkeddatafragments.org/publications/jws2016.pdf.

[121] Sixuan Wang and Gabriel Wainer. “Modeling and simulation as a service architecture for deploying resources in the Cloud”. In: *International Journal of Modeling, Simulation, and Scientific Computing* 07.01 (Mar. 2016). DOI: 10.1142/s1793962316410026.

[122] Marcus Wiens, Tobias Meyer and Philipp 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, Sept. 2021. DOI: 10.3384/ecp21181235.

[123] Adam Wiggins. *The Twelve-Factor App*. 12th July 2023. URL: https://12factor.net/ (visited on 27/08/2023).

[124] Wikipedia contributors. *Internet*. In: *Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Internet&oldid=1179338793 (visited on 18/10/2023).

[125] Mark D. Wilkinson et al. “The FAIR Guiding Principles for scientific data management and stewardship”. In: *Scientific Data* 3.1 (Mar. 2016). DOI: 10.1038/sdata.2016.18. URL: https://www.nature.com/articles/sdata201618.

[126] Bernard P. Zeigler, Saurabh Mittal and Mamadou Kaba Traore. “MBSE with/out Simulation: State of the Art and Way Forward”. In: *Systems* 6.40 (2018). ISSN: 2079-8954. DOI: 10.3390/systems6040040. URL: http://www.mdpi.com/2079-8954/6/4/40.

[127] Chaim Zins. “Conceptual approaches for defining data, information, and knowledge”. In: *Journal of the American Society for Information Science and Technology* 58.4 (2007), pp. 479–493. ISSN: 1532-2890. DOI: 10.1002/asi.20508.

[128] Khaldoon Al-Zoubi and Gabriel Wainer. “Distributed Simulation Using RESTful Interoperability Simulation Environment (RISE) Middleware”. In: *Intelligence-Based Systems Engineering*. Ed. by Andreas Tolk and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–157. ISBN: 978-3-642-17931-0. DOI: 10.1007/978-3-642-17931-0_6.

A. Supplemental Content

A glossary of the core technical terms used in this thesis is provided in table A.1 (the explanations aim to be understandable rather than precise). In table A.2, table A.3 and table A.4, details on the evaluation of the developed solutions with respect to FAIRness and coupling are documented. In these tables, the term *FMU* is used to denote a .fmu-file; *HTTP-API* represents the non-RESTful HTTP-API; and *hypermedia API* represents the M&S-capabilities exposed through the hypermedia API.

The content of this appendix was originally published in [94].

A.1 Glossary

Term	Explanation
CNA	cloud-native application — applications that are specifically designed such that they exhibit the characteristics of cloud computing, such as on-demand self-service, measured service, pay-as-you-go, horizontal scalability et cetera [52]
FAIR	Findable, Accessible, Interoperable, Reusable — technology-independent guiding principles for data publishing with the intent to maximize accessibility and reuse, both for humans and machines [125]
FMI	Functional Mock-up Interface — open standard for the exchange and co-simulation of dynamic system models [65]
FMU	Functional Mock-up Unit — model exported according to the FMI standard
generic software agent	software that solves tasks that it has not been programmed for at a syntactic level [16]
HATEOAS	hypermedia as the engine of application state — essential constraint of the architectural style REST, roughly summarized as “client selection of options provided by service in-band” [118, sec. 3.3.4]
hypermedia API	application programming interfaces for software clients that are accessible over the internet and fully implement the REST constraints [117, p. 276]
MSaaS	Modelling and Simulation as a Service — umbrella term for efforts attempting to make M&S capabilities available as a service
PPA	Pragmatic Proof Algorithm — algorithm that can compose and execute hypermedia APIs for which RESTdesc descriptions exist [119]
RDF	Resource Description Framework — distributed data model [57]
REST	Representational State Transfer — architectural style underlying the Web [118]
RESTdesc	format for describing which transitions are possible in a given application state and what the effects of these transitions in terms of changes to the shared state are [119, sec. 4.3]
SOA	service-oriented architecture — architectural style for creating manageable, large-scale distributed applications [75]
TPF	Triple Pattern Fragment — query interface for RDF data; triples only [120]
QPF	Quad Pattern Fragment — query interface for RDF data; quads supported

Table A.1: Glossary

A.2 Details on the Evaluation of FAIRness and Loose Coupling

FAIR Principle	Interface	Value	Reason
F1. (Meta)data are assigned a globally unique and persistent identifier.	FMU	2	supported via field `guid` in `modelDescription.xml` (inside .fmu-file)
	HTTP-API	3	URLs globally unique; persistence not guaranteed but possible
	hypermedia API	3	URLs globally unique; persistence not guaranteed but possible
F2. Data are described with rich metadata (defined by R1 below).	FMU	3	basic metadata specified, more possible via vendor annotations
	HTTP-API	3	part of resource representations
	hypermedia API	3	part of resource representations
F3. Metadata clearly and explicitly include the identifier of the data they describe.	FMU	4	metadata within `modelDescription.xml` clearly about the FMU
	HTTP-API	1	not possible to explicitly link metadata to data; only through hierarchy
	hypermedia API	4	ensured by use of RDF
F4. (Meta)data are registered or indexed in a searchable resource.	FMU	2	(meta)data could be indexed by crawlers
	HTTP-API	2	(meta)data could be indexed by crawlers
	hypermedia API	3	indexation possible; searchable via TPF interface
A1. (Meta)data are retrievable by their identifier using a standardized communications protocol.	FMU	1	not retrievable through identifier
	HTTP-API	3	documents containing (meta)data can be resolved via HTTP
	hypermedia API	4	all URLs resolve via HTTP
A1.1 The protocol is open, free, and universally implementable.	FMU	1	—
	HTTP-API	4	HTTP(S) is open, free, universally implementable
	hypermedia API	4	HTTP(S) is open, free, universally implementable
A1.2 The protocol allows for an authentication and authorization procedure, where necessary.	FMU	1	—
	HTTP-API	3	supported by HTTP(S), currently not used
	hypermedia API	3	supported by HTTP(S), currently not used
A2. Metadata are accessible, even when the data are no longer available.	FMU	1	—
	HTTP-API	1	metadata in JSON representations only
	hypermedia API	2	currently not implemented; organizational aspect

Table A.2: Details on the evaluation regarding the FAIR principles: aspects Findability, Accessibility

FAIR Principle	Interface	Value	Reason
I1. (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.	FMU	1	not supported
	HTTP-API	1	not supported
	hypermedia API	4	RDF/OWL/SHACL are W3C recommendations
I2. (Meta)data use vocabularies that follow FAIR principles.	FMU	1	—
	HTTP-API	1	—
	hypermedia API	3	reuse of well-known ontologies; evaluated via FOOPS! scanner for new ones
I3. (Meta)data include qualified references to other (meta)data.	FMU	1	not supported
	HTTP-API	1	not supported
	hypermedia API	2	supported, partly user input; only exemplary triples implemented
R1. (Meta)data are richly described with a plurality of accurate and relevant attributes.	FMU	2	supported via vendor annotations
	HTTP-API	2	theoretically possible (requires programming)
	hypermedia API	3	partly implemented; ambiguous; more to be added
R1.1 (Meta)data are released with a clear and accessible data usage licence.	FMU	2	supported (field in `modelDescription.xml`)
	HTTP-API	2	possible, requires programming
	hypermedia API	3	work in progress
R1.2 (Meta)data are associated with detailed provenance.	FMU	1	maybe through vendor annotations?
	HTTP-API	1	not supported
	hypermedia API	3	work in progress; not yet automated
R1.3 (Meta)data meet domain-relevant community standards.	FMU	4	FMI *is* the community standard
	HTTP-API	4	FMI *is* the community standard; FMU can be downloaded
	hypermedia API	4	FMI *is* the community standard; FMU can be downloaded

Table A.3: Details on the evaluation regarding the FAIR principles: aspects Interoperability, Reuse

Coupling Facet	Interface	Value	Reason
Discovery	HTTP-API	3	*referral* supported
	hypermedia API	3	*referral* supported
Identification/Naming	HTTP-API	3	*global*; via URLs
	hypermedia API	3	*global*; via URLs
Binding	HTTP-API	1	*early*; against OpenAPI Specification
	hypermedia API	3	*late*; through HATEOAS
Platform	HTTP-API	3	*independent*
	hypermedia API	3	*independent*
Interaction	HTTP-API	1	*synchronous*; client and server must be online
	hypermedia API	1	*synchronous*; client and server must be online
Interface Orientation	HTTP-API	1	*horizontal*; hardcoded requests
	hypermedia API	3	*vertical*; requests generated at run-time
Data Model	HTTP-API	1	*application-specific*; communicated via OAS
	hypermedia API	3	*self-descriptive*: RDF, OWL, SHACL
Granularity	HTTP-API	2	depends on what part of the API is used
	hypermedia API	2	both: resource representations fine, TPF coarse
State	HTTP-API	3	*stateless* messages
	hypermedia API	3	*stateless* messages
Evolution	HTTP-API	2	depends on programmers
	hypermedia API	3	*compatible* through self-descriptiveness/late binding
Generated Code	HTTP-API	1	*static*; against OAS
	hypermedia API	3	*dynamic*; at run-time
Conversation	HTTP-API	1	*explicit*; hardcoded at design-time
	hypermedia API	3	*reflective* → Pragmatic Proof Algorithm

Table A.4: Details on the evaluation regarding the coupling facets according to [76]

B. Publications and Software

This appendix provides annotated lists of the publications and software created as part of the work presented in this thesis.

B.1 Publications Related to This Thesis

First, the scientific publications are listed in reverse chronological order and their individual focus as well as their interrelations are outlined.

[94]: Journal article in *Semantic Web—Interoperability, Usability, Applications*[1] (impact factor 3.105) by IOS Press.
The article presents the motivation for and design concept of the developed M&S hypermedia API in detail; describes its implementation with a focus on resource modelling and advertising service capabilities; presents the proposed extension to the PPA and evaluates the achieved FAIRness and coupling characteristics.

Moritz Stüber and Georg Frey. “Dynamic system models and their simulation in the Semantic Web”. In: *Semantic Web Journal* (June 2023). Ed. by Bahar Aameri et al., pp. 1–36. DOI: 10.3233/sw-233359

[95]: Article in *atp magazin*[2] by Vulkan-Verlag GmbH (in German).
The article argues that the FAIR principles should be applied to the domain of M&S and why. The architecture of the hypermedia API is outlined, and it is shown that the FAIRness of M&S entities and capabilities increases iff they are exposed through this API. Neither intelligent software agents such as the PPA or coupling characteristics are discussed. *This article is a revised and slightly extended version of [96].*

Moritz Stüber and Georg Frey. “FAIRness für Modelle und Simulationen”. In: *atp magazin* 63.10 (Oct. 2022), pp. 58–65. DOI: 10.17560/atp.v63i10.2626

[96]: Article in the proceedings of the conference *AUTOMATION 2022* published by VDI Wissensforum GmbH (in German). The contents were presented at the conference in Baden-Baden/Germany. *The article was published in revised and extended form as [95].*

Moritz Stüber and Georg Frey. “FAIRness in der Automatisierungstechnik am Beispiel Modellierung und Simulation”. In: *Automation 2022*. VDI Verlag, 2022, pp. 343–354. DOI: 10.51202/9783181023990-343

[1] https://www.iospress.com/catalog/journals/semantic-web
[2] https://atpinfo.de/atp-magazin/

[93]: Article in the proceedings of the 14th Modelica Conference 2021 published by Linköping University Electronic Press. The contents were presented at the conference (online).
This article describes the cloud-native, non-RESTful HTTP-API that preceded the M&S hypermedia API. Concepts, software architecture and limitations as well as two exemplary applications and related work in the Modelica-community are presented.

Moritz Stüber and Georg Frey. "A Cloud-native Implementation of the Simulation as a Service-Concept Based on FMI". in: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Linköping University Electronic Press, Sept. 2021. DOI: 10.3384/ecp21181393

[97]: Journal article in the "Smart Grids" section of *Frontiers in Energy Research*[1] (impact factor 3.858).
This article presents an investigation of the forecast quality achieved for predicting the power output of a small-scale PV system using both a physics-based and a data-driven approach. The non-RESTful HTTP-API was used for the simulations of the physics-based models, which represents another application of the developed software for simulating models in the cloud using requests programmed at design-time (compare section 6.2.1). Other than that there is no connection to the line of work presented in this thesis.
The paper is the result of a collaboration with authors working at the German Research Center for Artificial Intelligence[2]. I implemented the physics-based PV performance model in Modelica, designed, implemented and carried out the data analysis using Python and pandas, and wrote the paper including all figures except section 3.2.

Moritz Stüber et al. "Forecast Quality of Physics-Based and Data-Driven PV Performance Models for a Small-Scale PV System". In: *Frontiers in Energy Research* 9 (2021), p. 108. ISSN: 2296-598X. DOI: 10.3389/fenrg.2021.639346. URL: https://www.frontiersin.org/article/10.3389/fenrg.2021.639346

[92]: Article in the proceedings of the 15th International Conference in Control, Automation and Robotics (ICINCO) 2018 published by SCITEPRESS. The contents were presented at the conference in Porto/Portugal.
The article describes the concept of exposing M&S *as a service* and outlines a possible use case (determination of the flexibility of a power-to-heat system).

Moritz Stüber, Lukas Exel and Georg Frey. "Using Modelling and Simulation as a Service (MSaaS) for Facilitating Flexibility-based Optimal Operation of Distribution Grids". In: *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO*. INSTICC. SciTePress, 2018, pp. 613–620. DOI: 10.5220/0006899106230630

[1] https://www.frontiersin.org/journals/energy-research/sections/smart-grids
[2] https://www.dfki.de/en/web/research/research-departments/cognitive-assistants

B.2 Developed Software

URL	Content	Language(s)
/UdSAES/fmi2rdf	FMI-ontology; `fmi2rdf`-parser	RDF(S); OWL; Python
/UdSAES/sms-ontology	SMS-ontology; SMS-FMI-ontology	RDF(S); OWL
/UdSAES/simaas-api	M&S hypermedia API (interface)	Node.js
/UdSAES/simaas-worker	M&S hypermedia API (worker)	Python
/UdSAES/simaas-demo	Examples ensemble forecast PV power generation, component selection via GA	Python
/UdSAES/pragmatic-proof-agent	PPA-implementation; example thumbnail generation	Python; Node.js
/UdSAES/pv-systems	Model library for calculating the power generated by PV systems based on irradiance in the horizontal plane, temperature, wind speed	Modelica

Table B.1: Overview on the developed software; all URLs are relative to `https://github.com`.