Dynamic System Models and their Simulation in the Semantic Web

Moritz Stüber [a,*] and Georg Frey [a]
[a] *Chair of Automation and Energy Systems, Saarland University, Germany*
E-mails: moritz.stueber@aut.uni-saarland.de, georg.frey@aut.uni-saarland.de

Abstract. Modelling and Simulation (M&S) are core tools for designing, analyzing and operating today's industrial systems. They often also represent both a valuable asset and a significant investment. Typically, their use is constrained to a software environment intended to be used by engineers on a single computer. However, the knowledge relevant to a task involving modelling and simulation is in general distributed in nature, even across organizational boundaries, and may be large in volume. Therefore, it is desirable to increase the FAIRness (Findability, Accessibility, Interoperability, and Reuse) of M&S capabilities; to enable their use in loosely coupled systems of systems; and to support their use by intelligent software agents. In this contribution, the suitability of Semantic Web technologies to achieve these goals is investigated and an open-source proof of concept-implementation based on the Functional Mock-up Interface (FMI) standard is presented. Specifically, models, model instances, and simulation results are exposed through a hypermedia API and an implementation of the Pragmatic Proof Algorithm (PPA) is used to successfully demonstrate the API's use by a generic software agent. The solution shows an increased degree of FAIRness and fully supports its use in loosely coupled systems. The FAIRness could be further improved by providing more "rich" (meta)data.

Keywords: Models and Simulation as a Service, FMI, Hypermedia API, Pragmatic Proof Algorithm, FAIR Principles

1. Introduction

In general, models allow inferring new information based on what is already known by means of reasoning or, in the case of formally described models, by computation. They are a useful tool for the design, analysis and understanding of complex systems.

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

Today's industrial systems are complex mechatronic systems, integrating mechanical, electrical and computational elements. In this context, formal models are mathematical relations describing the dynamic system behaviour. 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 the system of DAEs by means of numerical integration algorithms is called simulation; it is the computational process by which a trajectory of values over time is retrieved as the result.

Both types of models—ontologies and systems of DAEs—are 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

*Corresponding author. E-mail: moritz.stueber@aut.uni-saarland.de.

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 this formalization, a limit in scope and expressivity and therefore a limit on the class of problems that can be solved using a certain language, including its ecosystem such as model libraries, Integrated Development Environments (IDEs) and expert communities, is imposed.

Naturally, this raises the question of whether two distinct modelling approaches and the corresponding ecosystems can be meaningfully combined in order to enlarge the class of problems that can be investigated, drawing on the respective strengths of the individual approaches and alleviating their disadvantages.

1.1. Models, Simulation and FAIRness

In engineering, models and simulations are ubiquitous and used in many steps of a product's lifecycle. For example, the feasibility of a design with respect to requirements, safe operating conditions, and the sizing of components can be evaluated using Modelling and Simulation (M&S). "What if?"-questions can be analyzed 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 analyze 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 paper, 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 as well as historical reasons, many different formalisms and corresponding ecosystems exist today. They range from the use of general-purpose programming languages (such as Python) via modelling languages explictly designed to support multi-domain models as well as language features that facilitate the development of robust, well-structured models (such as Modelica) to highly specialized languages and tools for specific applications.

Unfortunately, models encoded in different languages are generally not interoperable, which is problematic for several reasons. From a practical perspective, the lack of interoperability hinders and slows down 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 this problem, the Functional Mock-up Interface (FMI) standard [1] 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 requires 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 .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]. Despite some inherent limitations that warrant the development of the upcoming version 3.0 of the standard [2], it is in general seen as a solution to the interoperability problems outlined above. However, interoperability is just one dimension of enabling reuse according to the Findable, Accessible, Interoperable, Reusable (FAIR) principles.

The FAIR principles [3] are a set of 15 guidelines intended to enable/facilitate the reuse of scholarly output such as models. They are listed in Table 3 and 4 in appendix A. The principles comprise technical and organizational aspects as well as guidelines on which facets of data and metadata to expose. Organizational aspects are aspects that require long-term commitment to the FAIRness of a data set, such as the use of persistent identifiers, the registration of (meta)data in a searchable resource, the continued existence of metadata even if the data is no longer available, and the use of FAIR vocabularies. The FAIR principles are formulated technology-independent and specifically target both human *and* software agents as the intended consumers of FAIR data. This focus on machine-actionability and the principles themselves suggest the Semantic Web technology stack as a suitable candidate for realizing FAIR digital assets; therefore, FAIRness is a relevant topic for the Semantic Web community.

Machine-actionability is seen as a varying degree of information provided to a software agent regarding an object's identity, applicability with respect to a goal, usability in terms of licensing and accessability, and usage instructions in a way that it enables the agent to take action [3, p. 3].

FAIR data is widely seen as desirable, and major research networks committed to supporting researchers in making their data FAIR [4]. Note that the principles do not imply making data available for free; rather, they emphasize the importance of enabling everyone to learn about the existence and content of digital assets, which may well be protected from public access for many reasons, including economical ones [5, p. 51]. This means that the FAIR principles can be as relevant for knowledge-driven academia as they can be for profit-oriented companies.

From our perspective, M&S suffers from a lack of FAIRness which limits the positive impact M&S could have. For example, consider the findability aspect: Models written in Modelica are organized in libraries. There is a list of libraries on the Modelica homepage[2] and a searchable index of libraries exists, but there is currently no way of searching for models that can represent a certain system other than opening a library which potentially contains it; searching for model names; and/or searching the documentation of the model library. Given a collection of FMUs, searching would be limited to the file name of the FMU and the contents of the descriptive .xml-file contained in the archive.

1.2. Coupling of Distributed M&S Capabilities

The knowledge required to solve complex engineering problems is distributed in nature: mechatronic systems are multi-disciplinary by definition; systems are developed by different persons or teams, often in parallel; and if components or machinery by other manufacturers are used, then the knowledge is also distributed across organizational boundaries.

Because of the distribution of knowledge across organizational boundaries, it is impossible to enforce a common format for knowledge exchange and the resulting situation can be characterized as an open world of diverse stakeholders. For connecting services in such situations, it is desirable to achieve *loose coupling* between the connected services [6, p. 919]. Loose coupling is a multi-faceted metric for designing systems of systems that aim to be robust, yet scalable by supporting the independent evolution of individual systems through minimizing the assumptions made about them [6].

The FAIR principles demand that "(Meta)data are retrievable by their identifier using a standardised communication protocol" (A1). This means that models must become available as part of a distributed system of systems. In turn, this means that they should be made available in a way that supports loose coupling.

[1] https://fmi-standard.org/tools/
[2] https://modelica.org/libraries

Just like the concepts and technology stack of the Semantic Web suggest themselves for making digital assets FAIR, the architectural style of the Web (Representational State Transfer (REST)) and its technology stack (Hypertext Transfer Protocol (HTTP), Uniform Resource Locators (URLs), hypermedia), suggest themselves for realizing loosely coupled systems because REST can be implemented such that loose coupling is fully supported [6, p. 919] and the Semantic Web was envisioned as an *extension* of the Web, suggesting that there should be no conceptual incompatibilities.

From a practical perspective, REST can be roughly summarized as follows (see [7] for a detailed explanation): a service exposes a set of conceptual resources such as "today's air temperature on campus". These resources are identified and located by URLs. As a reaction to the application of HTTP verbs (`GET`, `POST`, ...), a *representation* of the resource, for example an Hypertext Markup Language (HTML) document rendered as a website by the browser that sent the request, is returned. 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 [8], such as links and forms. This constraint is called Hypermedia As The Engine Of Application State (HATEOAS), and humans make use of this principle successfully everyday when browsing the Web to achieve their goals without first reading documentation on how to navigate a website.

However, the audience for FAIR data includes software agents because the amount of data available becomes increasingly overwhelming for humans, who cannot process the data at the same speed and volume as machines [3, p. 3].

On today's Web, software clients get access to functionality via Web Application Programming Interfaces (APIs). These APIs are often *based on* REST such that they expose resources of which JavaScript Object Notation (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*, which is neither RESTful (even though such APIs often denote themselves as such), nor fully supports loose coupling.

The programming of clients against a static service interface description at design-time is especially problematic when exposing M&S capabilities 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 [9, 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.

To summarize, Modelling and Simulation exhibit a lack of FAIRness and, consequently, machine-actionability that keep it from reaching its potential. For realizing software that exposes M&S capabilities FAIRly, it is desirable to support loose coupling.

1.3. Research Questions and -Hypotheses

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 on the Web to generic software agents [10].

The meaning of things is expressed using the Resource Description Framework (RDF) data model, which represents data as graphs of nodes connected by directed edges: a *subject* node is connected to the *object* node by a *predicate*. Different serializations of the resulting subject–predicate–object *triples* (or subject–predicate–object–graph *quads*) exist. Interfaces to RDF data range from the ultimate expressivity of SPARQL Protocol and RDF Query Language (SPARQL) endpoints (which can be expensive for the server) to the simplicity of downloading data dumps of an entire data set (which contradicts the idea of using data within the Semantic Web).

APIs for software clients that are accessible over the internet and fully implement the REST constraints are called *hypermedia APIs* [11, p. 276]. Consequently, a hypermedia API that uses the RDF data model for its resource representations 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, it is not restricted to read-only access as all HTTP methods can theoretically be supported.

What are generic or intelligent software agents, though? Cardoso and Ferrando define intelligent agents 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)" [12]. Kirrane and Decker [13] 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. Moreover, they call for basing the development of intelligent agents on the FAIR principles since they also see a "strong connection between said principles and Semantic Web technologies and Linked Data principles" [13, p. 3].

One task that needs to be solved by intelligent agents is to figure out which requests to send in which sequence in order to reach a goal, given a set of hypermedia APIs. This is precisely the purpose of the Pragmatic Proof Algorithm (PPA) published by Verborgh et al. [14]; therefore, the PPA can be seen as an intelligent software agent. However, there was no Free/Libre and Open-source Software (FLOSS) implementation of the PPA available, which is why we implemented it based on the information given in [14].

We see hypermedia APIs, as an exemplary specific interface to RDF data, and the PPA, as an example of a generic software agent, as promising candidates to improve the FAIRness of M&S capabilities in way that supports loose coupling. This idea raises three questions:

Q1. Can the FAIRness of M&S 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?

From these research questions follow four hypotheses:

H1. *In combination, the developed M&S hypermedia API and the implementation of the PPA allow both human and software agents to solve tasks involving models and their simulation.* Compared to a REST-based Modelling and Simulation as a Service (MSaaS)-implementation, the solution is H1.1) more flexible and more robust against changes. Moreover, it H1.2) allows a declarative problem formulation.
H2. *Software agents can* autonomously *use the M&S hypermedia API to a) discover the exposed capabilities and determine the achievability of their goal; as well as b) query a collection of models and model instances; add, instantiate and simulate models; and retrieve the simulation results in a serialization of RDF.* Compared to a collection of FMUs and compared to a REST-based MSaaS-implementation, the M&S hypermedia API H2.1) increases the FAIRness and H2.2) machine-actionability of capabilities and also H2.3) supports its use in loosely coupled systems.
H3. *Researchers and software engineers can use the implementation of the PPA to achieve declaratively formulated goals by using any RESTdesc-enabled hypermedia API, including those that rely on graphs with a specific shape as input during interaction.* Moreover, they can review the code and use it to build their own applications.
H4. *Researchers and software engineers can use the developed ontologies, as well as the software developed to extract information from FMUs using these ontologies, to express information about FMUs in RDF; to describe essential M&S-concepts and their interrelations in RDF; and to reason about FMUs as well as systems, models, and simulations.* Compared to manually created, application-specific Knowledge Graphs (KGs), the solution H4.1) speeds up KG creation given FMUs and H4.2) facilitates integration of models, model instances, simulation specifications and -results with other linked data.

The first and second hypothesis can be seen as the core hypotheses of this work, whereas the third hypothesis mostly concerns the necessary *implementation* of the PPA and the fourth hypothesis follows from the second as the ontologies and parser are necessary building blocks to enable the creation of the hypermedia API.

For the second hypothesis, machine-actionability will be demonstrated through the API's use by the PPA-implementation as a software that was not specifically programmed to use it. FAIRness and support for loose coupling will be evaluated by comparing the developed hypermedia API to its non-RESTful predecessor (detailed in [9]) for each of the 15 FAIR principles and for each of the coupling facets identified by Pautasso and Wilde [3], respectively.

1.4. Outline

The remainder of this paper is structured as follows: first, related work on the combination of Semantic Web concepts and -technologies with the domain of Modelling and Simulation is summarized in section 2. Then, design and implementation of the software that is necessary to answer the research questions is described in section 3, followed by outlining two exemplary applications of the developed software in section 4. Next, it is analyzed whether or not the hypotheses could be validated; and characteristics and limitations of the approach are discussed in section 5. Last, section 6 summarizes and draws conclusions from the presented work.

2. Related Work

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 Model-based Systems Engineering (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.

2.1. Ontologies for Model-based Systems Engineering

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 [15].

With respect to *what* is modelled by an ontology, *methodological* and *referential* ontologies can be distinguished. 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") [15, 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 [15, pp. 136, 143].

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 [15, 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 [16, p. 68 f.].

Successful applications of ontologies in conjunction with M&S have been reported in three main categories:

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" [17, 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[3]. There is no specification directly targeting M&S. OSLC has a strong focus on human end-users, as for example shown through the 'resource preview' [18] and 'delegated dialogs' [19] features.

El-khoury 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 [20]. The author concludes that the software architecture of OSLC allows for scalable, decentralised solutions in a heterogeneous 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 [20, p. 25 f.]. Consequently, OSLC is seen as useful for traceing lifecycle information such as requirements across applications. The creation of the links that encode this trace, facilitated through delegated dialogs and resource preview, is identified as the functionality implemented most [20, tbl. 7, p. 25].

König et al. 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 [21]. 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 [21, p. 176].

Interoperability and Composability *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) [22, p. 6]. For the combination of models, the highest level of interoperability according to the LCIM, conceptual interoperability, is required because 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 [22, p. 7].

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" [15, p. 142], but point out that difficulties are expected for non-technical systems. Axelsson 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 [23].

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 [16]. 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 traceing 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.

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

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. 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) [24]. 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 [25]. Both base their implementations on FMI as the format for the component models and the System Structure and Parameterization (SSP) standard [26] 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 et al. point out that any language is insufficient for representing reality, and that the meaning of relations cannot always be grounded in logic [15, pp. 139–141]. Second, the descriptive and normative nature of ontologies need to be balanced, which is expected to be especially difficult for non-technical systems [15, 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 [27, p. 134].

2.2. 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.

First, Bell et al. [28] 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) [29] 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" [30]. 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. System Design and -Implementation

Attempting to answer the research questions requires the implementation of four main pieces of software: first, ontologies that allow the description of FMUs and M&S entities and -capabilities in RDF are required. Second, these descriptions should be generated automatically as far as possible, starting from the FMUs used. Third, a hypermedia API that exposes M&S capabilities in RDF using the developed ontologies in combination with established ones needs to be implemented. Last, an implementation of the PPA as a means to demonstrate the machine-actionability of the hypermedia API is required.

From a non-functional perspective, the developed software should be seen as a proof of concept because neither an explicit analysis of risks to its dependability and security, nor optimizations of any kind were performed. However,

best practices regarding software development and -operations (DevOps) were followed to a large extent and core functionality is tested through unit- and API tests.

The technology stack chosen for realization was selected according to the following criteria: first, only FLOSS components were considered that, when used, avoid implementing functionality that already has stable implementations. Second, the components are required to represent the state of the art and exhibit features that indicate high quality, such as useful documentation and/or a high number of users.

3.1. FMI to RDF

The essential first step to providing content in the Semantic Web is to gain the ability to "talk" about a domain of interest. Therefore, three ontologies were developed using the Protégé editor [31]: 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.

The FMI-ontology is essentially a transcription of definitions in the FMI standard document [1] 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.

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 also of type `ModelParameter` and belongs to the `Virtual` realm, as opposed to the `RealOrEnvisioned` realm, in which a `SystemProperty` is represented by the `ModelParameter`. The RDF descriptions enabled through the use of the SMS-ontology are independent of any specific modelling formalism such as FMI.

Note that neither the FMI-ontology nor the SMS-ontology define individuals, as these are intended to be created as part of a Knowledge Graph or application such as an instance of the M&S hypermedia API. A fictional excerpt of such a KG is shown in Figure 1 with the intent to visualize the main concepts and roles of the FMI- and SMS-ontologies.

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.

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. The representation that `fmi2rdf` creates includes Shapes Constraint Language (SHACL) shapes graphs that specify the requirements for instantiating and simulating a model in terms of the requirements for a RDF graph that contains the required parameter- and input values. SHACL [32] was chosen for encoding constraints on graphs because it is recommended by the W3C.

The `fmi2rdf`-parser is implemented in Python, using `FMPy`[4] for reading FMU properties and `rdflib`[5] for representing and serializing the graph built. It can be used through a Command Line Interface (CLI) as well as from

[4] https://github.com/CATIA-Systems/FMPy
[5] https://rdflib.readthedocs.io/en/stable/

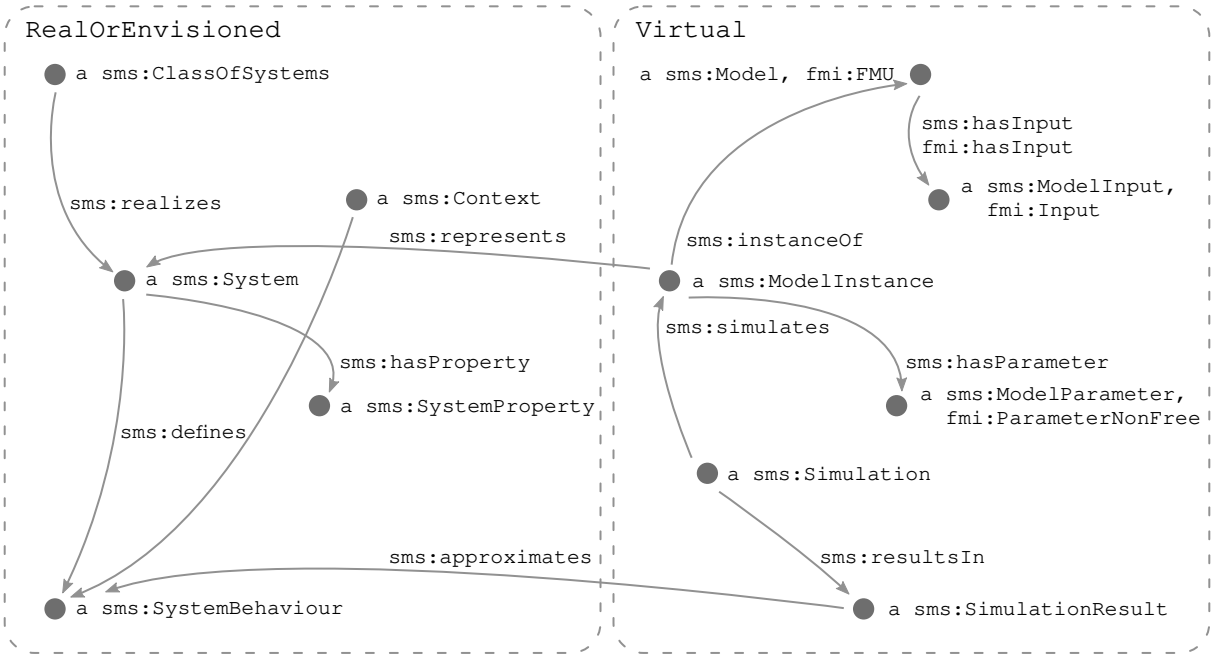

Fig. 1. The FMI- and SMS-ontologies allow relating abstract entities of the M&S-domain to their counterparts in the real (or envisioned) world, as shown by this graph visualization.

Table 1
Persistent URLs and repositories for the developed ontologies and the `fmi2rdf`-parser

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

Python code and is released under the MIT license on GitHub. Similarly, the ontologies are also released under the MIT license on GitHub; find the links in Table 1.

Note that the ontologies were given persistent URLs via the PURL service[6], which establishes a redirect currently pointing to the serialization of the ontology on the main branch in the GitHub-repository.

3.2. M&S hypermedia API

The M&S hypermedia API is an evolution of the "Cloud-native Implementation of the *Simulation as a Service*-Concept Based on FMI", which was presented at the Modelica conference 2021 [9]. This earlier version was a REST-based HTTP-API that used JSON-serializations for resource representations. It did not support HATEOAS and required programmers to code clients against the OpenAPI Specification, which was regenerated for every model added. The article [9] details concepts, implementation principles and limitations alongside the presentation of two exemplary applications and a discussion of related work in the Modelica community.

[6]https://purl.archive.org/help

Table 2
Overview of the service interface in terms of HTTP methods, exposed resources and their meaningful combinations (incomplete)

Method	Resource	Description
`POST`	/models	Add a new model to the API-instance
`GET`	/models/{model-id}	Retrieve a model representation from the API
`DELETE`	/models/{model-id}	Delete a model representation from the API
`POST`	/models/{model-id}/instances	Instantiate a model for a specific system
`GET`	/models/{model-id}/instances/{instance-id}	Get a representation of a specific model instance
`POST`	/models/{model-id}/instances/{instance-id}/experiments	Trigger the simulation of a model instance by defining an experiment
`GET`	/models/{model-id}/instances/{instance-id}/experiments/{experiment-id}	Retrieve a representation of a specific experiment definition and its status
`GET`	/models/{model-id}/instances/{instance-id}/experiments/{experiment-id}/result	Retrieve a representation of the results of a specific simulation run
`GET`	/knowledge-graph?subject=...&predicate=...&object=...&graph=...	Query API-instance via Triple Pattern Fragment-interface
`OPTIONS`	*	Retrieve RESTdesc descriptions in N3 serialization

Here, we briefly summarize the main ideas underlying the API's interface and software architecture and then focus on the aspects specific to turning the REST-based HTTP-API into a hypermedia API: resource modelling and the advertising of service capabilities.

Seen from a high-level point of view, the tasks to be solved by a M&S hypermedia API are

- to expose entities and functionality of the application domain in terms of uniquely identifiable conceptual resources which form the service interface;
- to advertise the service functionality and -interface in a machine-actionable manner; and
- to support querying for information that a service instance has.

In Table 2, the main part of the service interface in terms of the exposed resources and possible actions in terms of applicable HTTP methods is summarized. Hypermedia representations of the exposed resources can be requested in RDF serializations that support named graphs, such as JSON-based Serialization for Linked Data (JSON-LD) [33]. The earlier, non-RESTful version of the API is still available through content-negotiation when requesting JSON representations and serves as a baseline for the evaluation of FAIRness and loose coupling in section 5. Its interface is described according to the OAS, which is provided at `/oas` and rendered as a human-readable web page at `/ui`.

The most basic resource exposed is a model. Models can be instantiated by setting the model parameters; model instances can be simulated by specifying the properties of a simulation, such as initial conditions, input time series, and solver settings. Adding a new simulation by `POST`ing its definition to the API instance triggers a simulation run; the results of which become accessible via a link in the representation of the simulation-resource once it is completed. Moreover, the original .fmu-file can be downloaded by asking for a binary representation of the model resource (media type `application/octet-stream`).

The implementation of this functionality is structured in four main parts, as shown in Figure 2: 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.

There are several desirable consequences of this separation of concerns. First, the computing power can be scaled by adding more worker 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: in our implementation, the API uses Node.js for efficient handling of parallel requests using

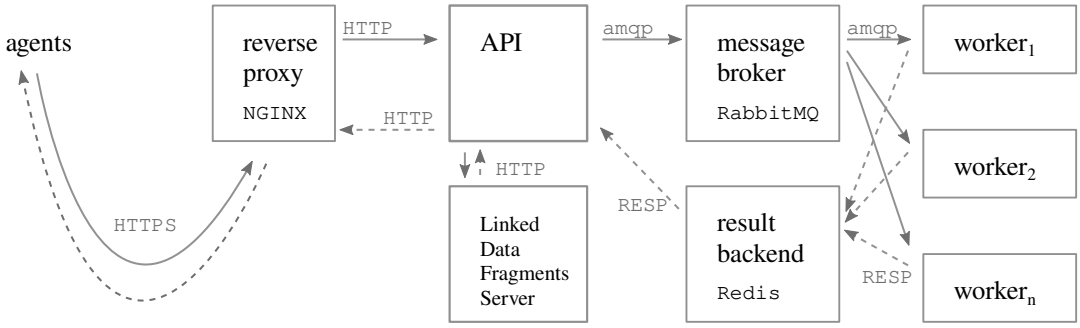

Fig. 2. The implementation is structured in distinct API- and worker components which exchange data via queues; a reverse proxy provides a HTTPS connection to users. An instance of https://github.com/LinkedDataFragments/Server.js enables querying (proxied through the API).

`promises` and the `async/await`-syntax, whereas the worker uses Python for handling FMUs via FMPy. API and workers are tightly coupled through the task- and result-representations exchanged via Celery[7] as the task queue implementation, relying on RabbitMQ[8] as the message broker and Redis[9] as the result backend.

API and worker are available under the MIT license at https://github.com/UdSAES/simaas-api and https://github.com/UdSAES/simaas-worker, respectively. From an operational perspective, all components are intended to be deployed on a clustered elastic platform such as Kubernetes and therefore support containers as deployment units. Please refer to the `README` documents for details on configuration and deployment.

3.2.1. Resource Modelling

Having decided on which resources to expose and which HTTP methods to allow on them, the question "which triples should the resource representation contain?" becomes the most important, yet also the most ambiguous question. It is important because it defines both the FAIRness and machine-actionability to a large extent—what isn't there can neither contribute to FAIR, nor be acted upon. It is also ambiguous because the FAIR principles do not offer much guidance on this except that there should be licensing (R1.1) and provenance (R1.2) information, that domain-relevant community standards should be met (R1.3), and that (meta)data should be "richly described with a plurality of accurate and relevant attributes" (R1). Designing an RDF representation with a specific application in mind may dictate which triples are needed, but it is a defining characteristic of both Service-oriented Architectures (SOAs) [34] and FAIRness that applications which may use a service are *not* known at design time.

The content of a resource representation can be categorized in data, metadata, context and controls [35]. *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 rely on past experience with similar websites, as humans do. Note that due to the ambiguouity of the question 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.

[7] https://github.com/celery/celery
[8] https://www.rabbitmq.com
[9] https://redis.io

As an example, consider the abbreviated TriG-serialization [36] of a model resource in Listing 1. The triples that encode that the resource identified by the Uniform Resource Identifier (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 as well as as data because they are part of the conceptual resource that is exposed.

To provide context and controls, an `<#about>`-graph is created that is explicitly linked to the default graph by the `foaf:primaryTopic`-relation (line 26). The reason for this is that 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 [35]. This problem is avoided if a serialization that supports quads is chosen and the data is put in the default graph and the other parts in dedicated graphs, as shown in Listing 1.

In the exemplary `<#about>`-graph, 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 [37] 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). 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.

Note that different ways of communicating controls are used. In the most simple form, links that are intended to be dereferenced by `GET` requests to a fully specified URL can be seen as controls, as for example the link to a research paper that was influenced by the resource in line 42. However, fully specified requests consist of an HTTP method, the request URL, headers, authorization information and possibly parameters in the body and/or path of the request. Therefore, alternatives such as `hydra:Operation`- and `http:Request`-specifications (used in Listing 2), possibly in combination with SHACL shapes graphs are required.

Also note that the specification of controls in a resource representation only partly enables HATEOAS; hints at the consequences of acting on a control are also necessary. The draft specification of the Hydra core vocabulary [37] points out that "Generally, a client decides whether to follow a link or not based on the link relation [...] which defines its semantics". Humans select among possibilities based on the label of the link; in the Semantic Web, the equivalent of a link's label is the predicate that relates a subject node to an object node. For link relations that are commonly used by different APIs, a common vocabulary such as Hydra core should be used. For example, links labelled `hydra:collection` are used to point to "collections somehow related to this resource" [37] in lines 36 and 37.

However, these links may not be specific enough for guiding a client through an application—for example, it may be necessary to distinguish between the collection of all models and the collection of all instances of a specific model. The `api:` namespace was created to define predicates required for making such distinctions and for providing additional required link relations. Since it is specific to the API, it was decided to expose this definition at `/vocabulary` as part of the hypermedia API.

3.2.2. Advertising Service Capabilities using RESTdesc

In order to decide whether or not using an API brings a client closer to reaching its goal, a concise description of the API's *functionality* is required, including the the effect of state-changing operations such as `POST` requests (which cannot be communicated using link relations alone). One format for explaining hypermedia API functionality in a machine-readable way are *RESTdesc descriptions* [11, 14].

RESTdesc descriptions are Notation3 (N3)[10] 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.

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

[10] https://www.w3.org/TeamSubmission/n3/

Listing 1: TriG-serialization of a model representation (abbreviated)

```
@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 2: RESTdesc description for retrieving a model representation

```
1  @prefix api: <http://example.com/vocabulary#> .
2  @prefix http: <http://www.w3.org/2011/http#> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix sh: <http://www.w3.org/ns/shacl#> .
5  @prefix sms: <https://purl.org/sms-ontology#> .
6
7  {
8      ?model rdf:type sms:Model .
9  }
10 =>
11 {
12     _:request http:methodName "GET" ;
13               http:requestURI ?model ;
14               http:headers [
15                   http:fieldName "Accept" ;
16                   http:fieldValue "application/trig"
17               ] ;
18               http:resp [ http:body ?model ] .
19
20     ?model api:allInstances _:allInstances .
21     ?model sms:instantiationShape _:instantiationShape .
22     _:instantiationShape rdf:type sh:NodeShape ; sh:targetNode _:parameterSet .
23 } .
```

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 model representation serialized in the TriG format shown in Listing 2. Line 8 states the precondition, followed by the description of the HTTP request in lines 12 to 18 that, if and only if (iff) successful, will result in the postcondition (lines 20 to 22). The RESTdesc description contains universally quantified variables, prefixed by a quotation 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 section 4.3 of [14].

In natural language, the content of Listing 2 reads as follows: "Given an URI denoting a `sms:Model` (line 8), 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 12 to 18). The model representation returned as a response will link the model to a node via an `api:allInstances` predicate and to another node through an `sms:instantiationShape` predicate (lines 20 to 21). 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 22)". 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 or not 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" [38, section 4.3.7]. 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 hypermedia API serves the RESTdesc rules for all relevant interations in one N3 document transferred as the response to an `OPTIONS` request at the path `*`.

3.2.3. Querying using Triple Pattern Fragments

So far, 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 inconvinient and inefficient.

Verborgh et al. developed *Triple Pattern Fragments (TPFs)* [39] 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 [40].

A TPF interface exposes all triples matching the pattern `?subject ?predicate ?object`, where all, none, or some of the terms can be specified. 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.

Clients can still use SPARQL to formulate their queries; however, the query needs to be decomposed into requests to the TPF endpoint and the results of these individual queries need to be combined to obtain the final result [39, 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 [39, 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 [39, p. 186]. Moreover, TPFs are compliant with REST and thus well suited for integration into a hypermedia API.

To support querying, a TPF 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`[11], to which the API proxies requests at the path `/knowledge-graph`.

3.3. Extended PPA-implementation

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

[11] https://github.com/LinkedDataFragments/Server.js

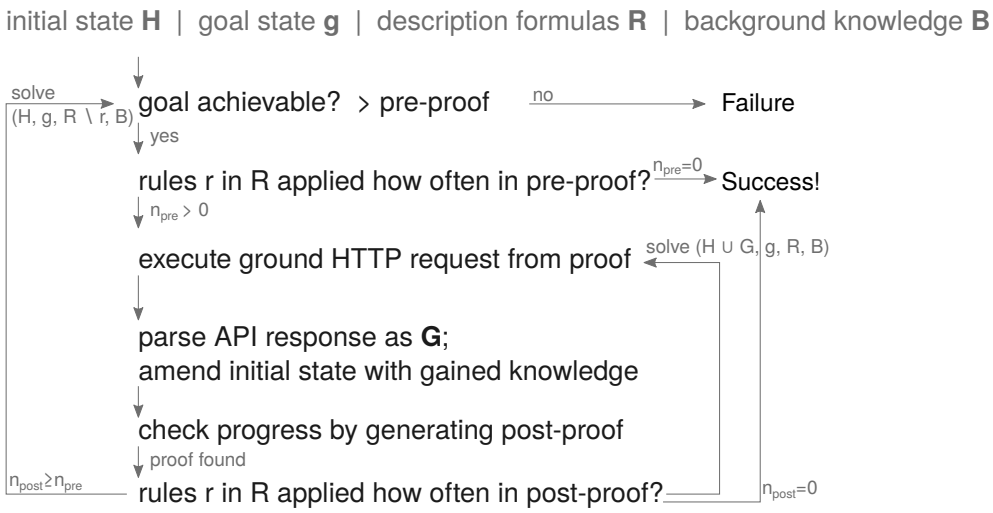

Fig. 3. The Pragmatic Proof Algorithm (PPA) [14] solves hypermedia API composition problems.

Proof Algorithm (PPA) by Verborgh et al. [14, p. 34] does. Moreover, it needs to be ensured that correctly shaped input data is provided iff necessary.

The PPA is visualized in Figure 3. 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. All of these are N3 formulas; see section 4 for examples. Together, they form an *API composition problem* [14, p. 22]. Then, 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 authors' 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 [41] is used for creating the necessary proofs. The source code for the PPA-implementation is released under the MIT license on GitHub at https://github.com/UdSAES/pragmatic-proof-agent.

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 run-time, 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 through a SHACL shape in the RESTdesc rule. As an example, consider lines 21 to 22 in Listing 2. 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 initially 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).

Iff 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.

4. Applications

In the abstract, the M&S hypermedia API gives access to two core functionalities: the simulation of model instances and retrieval of the simulation results; as well as the provisioning of models and model instances including the ability to query the data available. Two concrete uses of these functionalities are presented as examples.

4.1. Having Software Agents Run Simulations

Assume that a forecast for the power produced by a specific photovoltaic (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 shown in Listing 3. 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 $\star$. Then, it is checked whether or not 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 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

Listing 3: A 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.

```
@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 .
} .
```

a `POST` request to `/models` is fully specified (no unknown universally quantified variables), so it is executed. The PPA learns that `<http://example.com/models/6157f34f> rdf:type sms:Model` through the response, so the precondition in Listing 2 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 `http://example.com/models/6157f34f` contains the definition of the shape `http://example.com/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

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

Before executing the third fully specified request, `POST /models/6157f34f/instances`, 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.

In conclusion, the example shows that it is now possible to *declaratively* instruct the PPA-implementation to add, instantiate and simulate a model and retrieve the simulation results. No programming is necessary, except for preparing input data if necessary. Furthermore, no static service interface description is needed; the description of possible state changing operations provided through RESTdesc in combination with the resource representations suffices.

4.2. Querying a Collection of Models

How does a potential user learn about the existence of models, though? Suppose one were interested in all models available that either represent the class of all PV systems or the class of all wind turbines. Furthermore, one wants to know for which specific systems in the real or envisioned world an instance of these models is used as a virtual representation. A possible implementation of this query in SPARQL is shown in Listing 4.

Listing 4: 'Which models represent PV systems or wind turbines in general, and which specific systems are represented by instances of these models?'

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

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

Since the M&S hypermedia API only exposes a TPF endpoint, a client is needed that decomposes the SPARQL query into a series of requests and executes them. For this, the `Comunica` framework [42] is used.

The ability to execute SPARQL requests against the M&S hypermedia API makes it a source of linked data which can be used in applications or to build KGs. However, these applications should consider that resources do not necessarily exist forever, either because they expire or because they are deleted by a user. These considerations are out of scope here, though.

5. Evaluation

In this section, it is stated how the claims were verified or falsified and the results are discussed. Then, the advantages and shortcomings of the current system design and -implementation are highlighted and the overall solution is compared against related work.

5.1. Method and Hypothesis Validation

Through the research questions and -hypotheses, three claims with respect to the M&S hypermedia API were made: that providing M&S capabilities through a RDF hypermedia API improves the FAIRness of the exposed resources; that the hypermedia API supports being part of a loosely coupled system of systems; and that it can be used by software agents that were not specifically programmed for the API's interface.

5.1.1. FAIRness (Q1, H2.1)

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

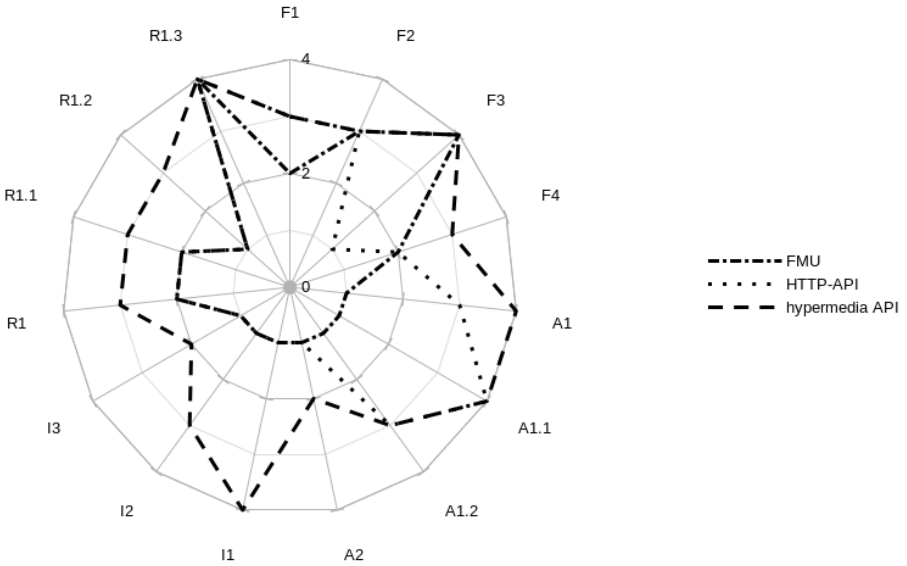

Fig. 4. The FAIRness of M&S resources increases when exposed through a hypermedia API both compared to a .fmu-file and a non-RESTful HTTP-API

'implemented'. The results of the evaluation are visualized in Figure 4. In the visualization, a higher value, plotted farther from the centre, corresponds to a higher degree of FAIRness.

Figure 4 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 resources 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 section 4.

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 3 and 4 in appendix A. 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 [43]. 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

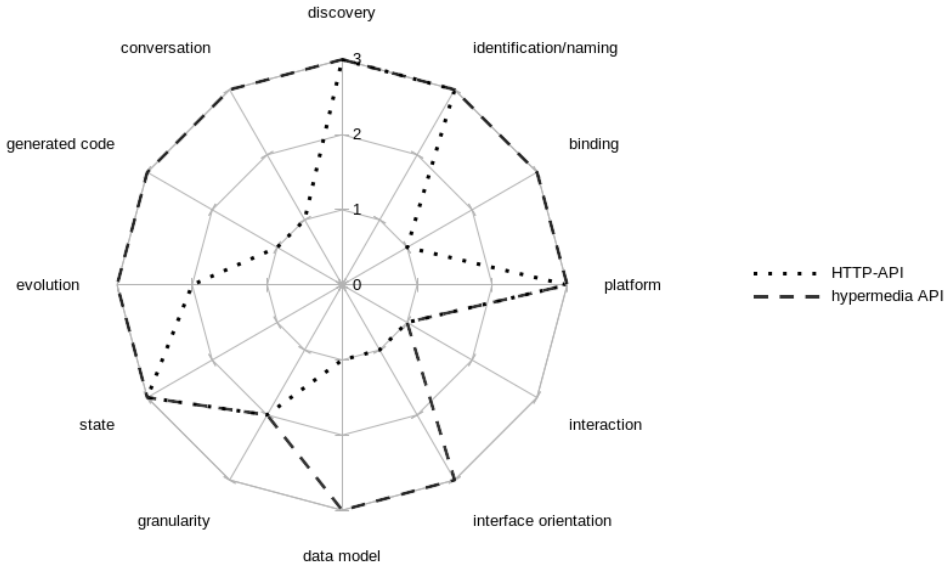

Fig. 5. In contrast to a non-RESTful HTTP-API, the M&S hypermedia API supports loose coupling according to all but one facet.

through organizational commitment and in-depth data modelling. Both are seen as beyond the scope of a proof of concept-implementation.

5.1.2. Machine-actionability (Q2, H2.2, H3)

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 4. This successful use is seen as an indication of increased machine-actionability (H2.2) as well as as verification of the third hypothesis (H3). 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.

5.1.3. Loose Coupling (Q3, H2.3)

Pautasso and Wilde [6] define loose coupling of service-oriented systems in terms of 12 facets and provide an exemplary analysis for RESTful, Simple Object Access Protocol (SOAP), and Remote Procedure Call (RPC) service interfaces. For each facet, they define what comprises *tight (numerical value 1)* and *loose (3)* coupling; for analysis, they suggest the additional class *design-specific (2)* in order to support the analysis of architectural styles and software architectures in addition to specific implementations.

The result of analysing the REST-based HTTP-API and the M&S hypermedia API with respect to coupling is visualized[12] in Figure 5. The detailed assessment can be found in Table 5 in appendix A.

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

[12]Note that the axis is reversed compared to the visualizations in [6] for better consistency within this paper.

(or operation signatures) to be exchanged within each interaction" and it is argued that fewer interactions through more coarse-grained interfaces results in more loosely coupled systems [6, p. 916]. The M&S hypermedia API offers both a relatively fine-granular interface through its choice of resources to be exposed as well as a coarse grained TPF 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 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 suggest that caching also decreases coupling with respect to the interaction-facet through *non-blocking* (but still synchronous) interaction [6, p. 915].

5.1.4. Hypotheses H1, H3, and H4

In the previous subsections, the validation of the second hypothesis was discussed in detail. From the validation of H2 follows that also the fourth hypothesis H4 must be valid: without the FMI- and SMS-ontologies, which are used by the `fmi2rdf` parser to generate RDF representations of FMUs added, the M&S hypermedia API would not be operational. The automatic creation of RDF representations of FMUs by `fmi2rdf` is certainly faster than a manual process (H4.1), and the resulting triples can be integrated into KGs and linked data applications (H4.2). The ontologies strive to reach a high degree of FAIRness and there was no FLOSS parser for representing FMUs in RDF available before.

As for the first hypothesis H1, the general functionality as well as the declarative problem formulation (H1.2) is shown through the successful application. The increased flexibility and robustness with regard to API changes (H1.1) is demonstrated using a variation of the image-resizing example used by Verborgh et al. to explain RESTdesc and the PPA [14]: as the base case, take a hypermedia API supporting RESTdesc which allows resizing an image by uploading the image in the body of a `POST` request to `/images`. The response to this request contains a `ex:smallThumbnail`-link at which the resized image can be downloaded via a `GET` request, for example `/images/9007eb1a/thumbnail`. Then, imagine that the paths at which the resources are exposed can be changed, for example to `/bilder/<id>/miniaturbild` (the German translation): requests hardcoded against one of the variants will fail with the other versions, while the PPA still works correctly regardless of the resources' path because it follows the HATEOAS principle. It is thus more robust against changes. An implementation of this is included in the repository https://github.com/UdSAES/pragmatic-proof-agent as an example. Its API tests only work for the English variant, whereas the PPA can also handle the German or French version.

The successful use of the image-resizing API by the PPA-implementation as well as the application presented in subsection 4.1 validate H3.

5.2. Discussion

The exposal of M&S capabilities through the developed hypermedia API exhibits several positive characteristics and potentials in addition to the increased FAIRness and support for loose coupling that are the focus of this work.

First, the use of RDF-serializations for resource representations enables the mitigation of trust issues and composability issues. Trust issues arise as a consequence of reusing content created by others: unless one has the ability and time to retrace and understand *every* detail of the reused entity, one has to trust that the entity does what it is supposed to do to a certain extent. One way of inspiring trust in a resource is to associate detailed provenance information with the resource, which should be generated automatically as far as possible (see for example [44]). Questions on the composability of two or more different models can be answered through the detailed, unambiguous description of models through RDF-based ontologies [15, p. 142]. Currently, neither these detailed model descriptions nor comprehensive provenance information are included in the resource representations of the presented M&S hypermedia API, but they are supported in principle and should be added in a future version.

Second, the M&S hypermedia API is designed as a Cloud-native Application (CNA) [45] and, consequently, as a Cloud-native Simulation (CNS) system [46, p. 15]: it is realized as a microservice, isolates the state in a single

component, uses containers as deployment units and follows best practices for the development of Software as a Service (SaaS) (for details see [9]). Thereby, horizontal scalability of the computing power is enabled.

Regarding the support for traceing 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.

Ontology-driven modelling on the base of the developed FMI- and SMS-ontologies represents an exciting opportunity for future work that has not been explored in detail yet.

Despite the positive aspects outlined above, there are many areas for improvement for the specific implementation of the M&S hypermedia API, as well as some conceptual shortcomings.

From a conceptual point of view, 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 user: 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.

From a practical perspective, the limitation to FMUs 2.0 for co-simulation with additional restrictions ([9, p. 397]); as well as the file-based storage of state in the API component and the corresponding lack of scalability of this component represent some of the areas for improvement in future versions. More details regarding opportunities for improving the implementation can be found in the 'Known Issues'-sections of the README-documents in each repository.

6. Conclusion

The work presented in this paper outlines one possible way to adress a core problem that prevents M&S resources and capabilities from reaching their potential in MBSE, namely the lack of FAIRness with respect to both human *and* software agents interested in the knowledge that the models encode. It was hypothesized that by exposing M&S capabilities through a hypermedia API that uses serializations of the RDF data model for its resource representations, the problem can be solved in a way that also supports loose coupling. Loose coupling is seen as a necessary characteristic because relevant data is in general distributed across organizational boundaries, meaning that adherence to a centralized model for data exchange as required by tightly coupled solutions cannot be enforced.

The implementation is based on the FMI standard for model exchange and co-simulation; specifically, the implementation currently only supports FMUs 2.0 for co-simulation with additional restrictions. To describe its resources, the M&S hypermedia API relies on the developed FMI- and SMS-ontologies in conjunction with the `fmi2rdf`-parser, which automatically generates RDF representations from FMUs.

The Pragmatic Proof Algorithm by Verborgh et al. was implemented as an example of a generic software agent that is capable of using hypermedia APIs that it has not been specifically programmed for. This is enabled by communicating the existence and envisioned consequences of state transitions supported by an API through RESTdesc descriptions. These descriptions are used for high-level planning on behalf of the PPA, but at run-time, the interaction is driven by client selection of service-provided options and thus follows the HATEOAS principle.

Finally, the querying of information about models and model instances is enabled through a TPF interface. This means that SPARQL queries can be evaluated against an instance of the M&S hypermedia API as long as the query engine supports the decomposition of the SPARQL query into a series of TPF requests.

The implemented software represents a proof of concept, but could nonetheless be directly useful for those with a collection of FMUs on which it should be possible to execute SPARQL queries; those wanting to execute a high number of standalone simulations on a horizontally scalable set of computing resources, as for example in parameter fitting applications; as well as to those who want to build upon the ideas and/or implementations presented in this paper.

To conclude, all hypotheses could be validated. The self-descriptiveness, data model, encoding format and retrieval protocol of the representations through which the M&S hypermedia API exposes its resources increases FAIRness and machine-actionability and also supports loose coupling. Researchers and software engineers are enabled to review and reuse the code because it is released publicly under the permissive MIT license.

Appendix A. Details on the Evaluation of FAIRness and Loose Coupling

In Tables 3, 4 and 5, details on the evaluation of the developed solutions with respect to FAIRness and coupling are documented. In all 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 presented in this paper.

Table 3
Details on the evaluation regarding the FAIR principles: aspects Findability, Accessibility, Interoperability

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 standardised 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 authorisation 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
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

References

[1] Modelica Association, Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0.2, Technical Report, Modelica Association, Linköping, 2020. https://fmi-standard.org.

[2] A. Junghanns, C. Gomes, C. Schulze, K. Schuch, P. R., M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson, T. Sommer, C. Bertsch, T. Blochwitz and M. Najafi, The Functional Mock-up Interface 3.0 - New Features Enabling New Applications, in: *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press, 2021. doi:10.3384/ecp2118117.

[3] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L.B. da Silva Santos, P.E. Bourne, J. Bouwman, A.J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C.T. Evelo, R. Finkers, A. Gonzalez-Beltran, A.J.G. Gray, P. Groth, C. Goble, J.S. Grethe, J. Heringa, P.A.C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S.J. Lusher, M.E. Martone, A. Mons, A.L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M.A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao and B. Mons, The FAIR Guiding Principles for scientific data management and stewardship, *Scientific Data* **3**(1) (2016). doi:10.1038/sdata.2016.18. https://www.nature.com/articles/sdata201618.

[4] Sorbonne declaration on research data rights, 2020. https://sorbonnedatadeclaration.eu/data-Sorbonne-declaration.pdf.

[5] B. Mons, C. Neylon, J. Velterop, M. Dumontier, L.O.B. da Silva Santos and M.D. Wilkinson, Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud, *Information Services & Use* **37**(1) (2017), 49–56. doi:10.3233/isu-170824. https://content.iospress.com/articles/information-services-and-use/isu824.

[6] C. Pautasso and E. Wilde, Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design, in: *18th International World Wide Web Conference*, 2009, pp. 911–920. http://www2009.eprints.org/92/.

[7] R. Verborgh, S. van Hooland, A.S. Cope, S. Chan, E. Mannens and R.V. de Walle, The fallacy of the multi-API culture: Conceptual and practical benefits of Representational State Transfer (REST), *Journal of Documentation* **71**(2) (2015), 233–252. doi:10.1108/JD-07-2013-0098.

[8] R.T. Fielding, REST APIs must be hypertext-driven, 2008. https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[9] M. Stüber and G. 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, 2021. doi:10.3384/ecp21181393.

[10] B. Glimm and H. Stuckenschmidt, 15 Years of Semantic Web: An Incomplete Survey, *KI – Künstliche Intelligenz* **30**(2) (2016), 117–130. doi:10.1007/s13218-016-0424-1.

[11] R. Verborgh, T. Steiner, R. Van de Walle and J. Gabarro, Linked Data and Linked APIs: Similarities, Differences, and Challenges, in: *The Semantic Web: ESWC 2012 Satellite Events*, E. Simperl, B. Norton, D. Mladenic, E. Della Valle, I. Fundulaki, A. Passant and R. Troncy, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 272–284. ISBN 978-3-662-46641-4.

[12] R.C. Cardoso and A. Ferrando, A Review of Agent-Based Programming for Multi-Agent Systems, *Computers* **10**(2) (2021). doi:10.3390/computers10020016. https://www.mdpi.com/2073-431X/10/2/16.

[13] S. Kirrane and S. 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*, R. Verborgh, T. Kuhn and T. Berners-Lee, eds, CEUR Workshop Proceedings, Vol. 2165, CEUR-WS.org, 2018. http://ceur-ws.org/Vol-2165/paper2.pdf.

[14] R. Verborgh, D. Arndt, S. Van Hoecke, J. De Roo, G. Mels, T. Steiner and J. Gabarró Vallés, The Pragmatic Proof: Hypermedia API Composition and Execution, *Theory and Practice of Logic Programming* **17**(1) (2017), 1–48. doi:10.1017/S1471068416000016. http://arxiv.org/pdf/1512.07780v1.pdf.

[15] M. Hofmann, J. Palii and G. Mihelcic, Epistemic and normative aspects of ontologies in modelling and simulation, *Journal of Simulation* **5**(3) (2011), 135–146. doi:10.1057/jos.2011.13.

[16] T. Tudorache, Employing Ontologies for an Improved Development Process in Collaborative Engineering, 2006.

[17] J. Amsden and A. Berezovskyi (eds), OSLC Core Version 3.0. Part 1: Overview, OASIS Standard, OASIS, 2021. https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/oslc-core.html.

[18] J. Amsden and A. Berezovskyi (eds), OSLC Core Version 3.0. Part 3: Resource Preview, OASIS Standard, OASIS, 2021. https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/resource-preview.html.

[19] J. Amsden and A. Berezovskyi (eds), OSLC Core Version 3.0. Part 4: Delegated Dialogs, OASIS Standard, OASIS, 2021. https://docs.oasis-open-projects.org/oslc-op/core/v3.0/os/dialogs.html.

[20] J. El-khoury, An Analysis of the OASIS OSLC Integration Standard, for a Cross-disciplinary Integrated Development Environment: Analysis of market penetration, performance and prospects, Technical Report, 978-91-7873-525-9, KTH, Mechatronics, 2020, QC 20200430. ISBN 978-91-7873-525-9. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-272834.

[21] C. König, A. Mengist, C. Gamble, J. Höll, K. Lausdahl, T. Bokhove, E. Brosse, O. Möller and A. Pop, 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, 2020. doi:10.3384/ecp20169168.

[22] A. 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, IEEE Computer Society, Washington, DC, USA, 2013, pp. 3–9. ISBN 978-0-7695-5138-8. doi:10.1109/DS-RT.2013.8.

[23] J. Axelsson, Achieving System-of-Systems Interoperability Levels Using Linked Data and Ontologies, *INCOSE International Symposium* **30**(1) (2020), 651–665. doi:10.1002/j.2334-5837.2020.00746.x.
[24] M. Mitterhofer, G.F. Schneider, S. Stratbücker and K. Sedlbauer, An FMI-enabled methodology for modular building performance simulation based on Semantic Web Technologies, *Building and Environment* **125** (2017), 49–59. doi:10.1016/j.buildenv.2017.08.021. http://www.sciencedirect.com/science/article/pii/S0360132317303736.
[25] M. Wiens, T. Meyer and P. Thomas, The Potential of FMI for the Development of Digital Twins for Large Modular Multi-Domain Systems, in: *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press, 2021. doi:10.3384/ecp21181235.
[26] Modelica Association, System Structure and Parameterization Version 1.0, Technical Report, Modelica Association, 2019. https://ssp-standard.org/.
[27] A. Tolk and J.A. Miller, Enhancing simulation composability and interoperability using conceptual/semantic/ontological models, *Journal of Simulation* **5**(3) (2011), 133–134. doi:10.1057/jos.2011.18.
[28] D. Bell, M. Lycett, S. de Cesare, S.J.E. Taylor and N. Mustafee, Service-oriented simulation using web ontology, *International Journal of Simulation and Process Modelling* **7**(3) (2012), 217–227. doi:10.1504/IJSPM.2012.049148.
[29] G.A. Silver, J.A. Miller, M. Hybinette, G. Baramidze and W.S. York, DeMO: An Ontology for Discrete-event Modeling and Simulation, *SIMULATION* **87**(9) (2011), 747–773, PMID: 22919114. doi:10.1177/0037549710386843.
[30] M. Tiller and D. 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, pp. 725–734. ISSN 1650-3740. doi:10.3384/ecp17132725.
[31] M.A. Musen, The Protégé Project: A Look Back and a Look Forward, *AI Matters* **1**(4) (2015), 4–12. doi:10.1145/2757001.2757003.
[32] D. Kontokostas and H. Knublauch, Shapes Constraint Language (SHACL), W3C Recommendation, W3C, 2017, https://www.w3.org/TR/2017/REC-shacl-20170720/.
[33] G. Kellogg, P.-A. Champin and D. Longley, JSON-LD 1.1, W3C Recommendation, W3C, 2020, https://www.w3.org/TR/2020/REC-json-ld11-20200716/.
[34] OASIS, Reference Model for Service Oriented Architecture 1.0, Technical Report, 2006, OASIS Open. http://docs.oasis-open.org/soa-rm/v1.0/.
[35] R. Verborgh, Turtles all the way down, 2015. https://ruben.verborgh.org/blog/2015/10/06/turtles-all-the-way-down/.
[36] A. Seaborne and G. Carothers, RDF 1.1 TriG, W3C Recommendation, W3C, 2014, https://www.w3.org/TR/2014/REC-trig-20140225/.
[37] M. Lanthaler, Hydra Core Vocabulary, Technical Report, 2021. http://www.hydra-cg.com/spec/latest/core/.
[38] R. Fielding and J. Reschke (eds), Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Technical Report, 2014. doi:10.17487/rfc7231.
[39] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck and P. Colpaert, Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web, *Journal of Web Semantics* **37–38** (2016), 184–206. doi:10.1016/j.websem.2016.03.003. http://linkeddatafragments.org/publications/jws2016.pdf.
[40] C. Buil-Aranda, A. Hogan, J. Umbrich and P.-Y. Vandenbussche, SPARQL Web-Querying Infrastructure: Ready for Action?, in: *The Semantic Web – ISWC 2013*, H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J.X. Parreira, L. Aroyo, N. Noy, C. Welty and K. Janowicz, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 277–293. ISBN 978-3-642-41338-4.
[41] R. Verborgh and J.D. Roo, Drawing Conclusions from Linked Data on the Web: The EYE Reasoner, *IEEE Softw.* **32**(3) (2015), 23–27. doi:10.1109/MS.2015.63.
[42] R. Taelman, J. Van Herwegen, M. Vander Sande and R. Verborgh, Comunica: a Modular SPARQL Query Engine for the Web, in: *Proceedings of the 17th International Semantic Web Conference*, 2018. https://comunica.github.io/Article-ISWC2018-Resource/.
[43] D. Garijo, O. Corcho and M. Poveda-Villalón, FOOPS!: An Ontology Pitfall Scanner for the FAIR Principles **2980** (2021). http://ceur-ws.org/Vol-2980/paper321.pdf.
[44] B.D. Meester, A. Dimou, R. Verborgh and E. Mannens, Detailed Provenance Capture of Data Processing, in: *Proceedings of the First Workshop on Enabling Open Semantic Science (SemSci)*, 2017, pp. 31–38. http://ceur-ws.org/Vol-1931/#paper-05.
[45] N. Kratzke and P.-C. Quint, Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study, *Journal of Systems and Software* **126** (2017), 1–16. doi:10.1016/j.jss.2017.01.001. http://www.sciencedirect.com/science/article/pii/S0164121217300018.
[46] N. Kratzke and R. Siegfried, Towards cloud-native simulations – lessons learned from the front-line of cloud computing, *The Journal of Defense Modeling and Simulation* **18** (2021), 39–58. doi:10.1177/1548512919895327.

Table 4
Details on the evaluation regarding the FAIR principles: aspect Reuse

FAIR Principle	Interface	Value	Reason
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 license.	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 5
Details on the evaluation regarding the coupling facets according to [6]

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* $\rightarrow$ Pragmatic Proof Algorithm