Environmental Modelling and Software 176 (2024) 106002

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft

The model-as-a-resource paradigm for geoscience digital ecosystems

Paolo Mazzetti [*], Stefano Nativi

Istituto sull'Inquinamento Atmosferico, CNR, Via Madonna del Piano 10, 50019, Sesto Fiorentino (FI), Italy

ARTICLE INFO

Handling Editor: Daniel P Ames

Keywords:
Scientific models
Interoperability
Digital ecosystems
MaaR
Earth observation

ABSTRACT

A long-term goal of environmental science and Earth observation is to enable the creation of a "Model Web" of semantically interconnected data and models. Geospatial models are usually exposed on the Web as services accessible through heterogeneous interfaces. However, such services, which represent instances of the paradigm called Model-as-a-Service (MaaS), cannot be easily exploited beyond their original use as defined by the service provider. To overcome this important limitation and better support transparency, reproducibility, replicability and reusability of the model (following the Open Science paradigm), we investigated the adoption of a Model-as-a-Resource (MaaR) approach, in which a model is considered a generic digital resource that, as such, can play different roles in different potential use cases. The proposed MaaR framework can play an important enabling role in the realization of those digital ecosystems that generate environmental knowledge. The main challenges and opportunities are discussed in the manuscript.

Software and data availability

The paper presents an architectural framework for model sharing which does not refer specifically to any software solution for implementation. Section §4.5 mentions a set of technologies (software and standards) that could be adopted for the implementation of the proposed framework. All the cited technologies are available as open standards or open source software from their publishers and developers. A couple of software solutions (DAB and VLAB) are developed by authors' research unit and served to implement proofs-of-concept for the proposed architectural framework. More detailed information on these technologies is provided below.

Name of the software	DAB (Discovery and Access Broker)
Developer	CNR-IIA
Contact information	enrico.boldrini@cnr.it
Programming language	Java
Cost	Free
Software availability	https://github.com/ESSI-Lab/DAB
License	GNU Affero General Public License v3.0
Name of the software	VLAB (Virtual Earth Laboratory)
Developer	CNR-IIA
Contact information	mattia.santoro@cnr.it
Programming language	Java
Cost	Free
Software availability	https://github.com/ESSI-Lab/DAB
License	GNU Affero General Public License v3.0

1. Introduction

1.1. Scientific computational models

Modelling is an essential activity for modern science. In particular, the so-called *representational* models can emulate the behavior of a well-delimited system providing useful insights on the world that surrounds us. They can come in many different fashions: scale models, analogical models, idealized models, toy models, etc. (Frigg et al., 2020). In the current scientific practice, mathematical models, which aim at providing a mathematical representation of a real system, are the most important ones. In late XIX century and early XX century, mathematics formalization was a major step that allows expressing mathematical proofs as a mechanical procedure or an algorithm. This development suggested that, if physical processes can be represented by mathematical formulas, and mathematical formulas can be encoded as algorithms, then some mechanical instrument might emulate physical processes. This dream became a reality with computers development and the introduction of the computer science. The advent of computers has transformed science and engineering. Based either on physical theories or on big data processing, scientific procedures have been implemented as software code and executed to simulate and predict the behavior of physical systems (Imbert et al., 2017). Then, computational models have

* Corresponding author.
E-mail address: paolo.mazzetti@cnr.it (P. Mazzetti).

https://doi.org/10.1016/j.envsoft.2024.106002
Received 28 September 2023; Received in revised form 20 February 2024; Accepted 27 February 2024
Available online 4 March 2024

been joined to represent more complex scenarios, in an integrated modelling approach (Laniak et al., 2013). More recently, the advancement of communication technologies allowed models to run on distributed infrastructures, improving their efficiency and scalability. Since the last decade of the 20th century, the World Wide Web (hereinafter the Web) has provided an infrastructure of easily accessible protocols and technologies that allows harmonized access to many different digital resources including scientific models and algorithms.

1.2. Open science

Science is recognized as our best source of knowledge not because it delivers truth, (which is an unrealistic expectation) but because it justifies its statements. In principle, every rational person can evaluate the grounding of a scientific result. Many aspects of what is recognized as a correct scientific method aim at assuring that scientific results can be controlled and evaluated (Barton et al., 2020). However, the complexity of current science methods puts in danger the evaluability of scientific results since only a minority of literate people can understand whether a mathematical equation, or a simulation code is a justification of a scientific result. The intricacy of big integrated models can be a barrier even for experts (Lloyd and Winsberg, 2018) while data-driven models pose even bigger challenges: although machine learning experiments can be precisely defined, leading to perfectly reproducible research (Braun et al., 2018), their black-box nature often makes their very meaning obscure.

These difficulties pose societal challenges related to the education system in modern societies and the mediating role of the scientific community, but do not exempt scientists from justifying their assertions.

The Open Science movement addresses this fundamental issue (Fecher et al., 2014) trying to overcome the cultural and technical barriers that obstacle the full documentation of scientific findings, as result of the adoption of new technologies and methodologies in science. According to the Open Science paradigm, every step in the generation of scientific knowledge should be transparent and evaluable. Open Science encompasses concepts like Open Data (sharing of data used for the generation of knowledge), Open Software (sharing of software used for data processing) and Open Access (to the results of scientific research also as source of further research).

Open Science advocates *transparency* of scientific results, enabling literate people to evaluate their soundness, *reproducibility*, that is the possibility to generate again the same scientific experiments, *replicability*, that is the possibility to adapt existing results to different contexts, and, finally, *reusability*, that is the possibility to utilize existing results for further scientific investigations.

This article investigates how data- and physics-driven computational models can benefit from appropriate use of the Web architecture to meet open science requirements, having in mind the application scenarios of green transition and sustainable development. The next section summarizes the role of software architectural styles in the design (and description) of distributed systems and their adoption in the Web environment. Section 3 4is central as it describes a potential multi-style framework for sharing scientific models treated as digital resources supporting high-level Open Science scenarios. Section 4 summarizes the scientific and technical constribution of the proposed framework. Section 5 discusses some important aspects related to the sharing of scientific models, in the more general context of the digital ecosystems' paradigm. Finally, section 6 draws some conclusions.

2. Background

To demonstrate the applicability of the Web architecture to scientific model sharing it is necessary to clarify the importance played by the different architectural styles. Therefore, it is essential to have a sufficient understanding of the characteristics that distinguish a service-oriented approach versus a resource-oriented one, in the Web environment. For this purpose, it is useful reconsider a set of well-established concepts in the light of model interoperability.

2.1. Software architectural styles for implementing distributed systems

Software architecture description mainly serves at system design and analysis to implement a system with some desired characteristics, and evaluate the features that distinguish an existing system, respectively. In technical literature, there are several possible definitions of software architecture (Lloyd and Winsberg, 2018). They all introduce some essential characteristics stemming from a couple of common and general concepts: a software architecture refers to a (computing) system and is expressed through a collection of structures – i.e., sets equipped with properties and relationships, and isomorphic to (the relevant part of) the target system. The architecture design expresses a set of constraints reducing the compatible structures.

In software engineering, another well-used concept is that of "architectural style". A well-known definition of architectural style (by the Web architect Roy Fielding) is the following: *An architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style* (Fielding, 2000). While a system architecture consists of the full specification of a target structure, an architectural style specifies only few high-level constraints, which are necessary to satisfy a set of more general features. Further constraints may differentiate among multiple architectures that are compliant with the same architectural style.

2.2. Resource-oriented and service-oriented software architectures

For Web distributed systems, the most widespread software architectures can be categorized (at the highest level) as *service-oriented* or *resource-oriented* architectures.

Service-oriented architectures are built around the concept of *service*[1] implying the idea of *acting on behalf of someone else* –i.e., a customer, a client, or a master. In 2006, the Organization for the Advancement of Structured Information Standards (OASIS) defined a Service Oriented Architecture (SOA) as: "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" (OASIS Open, 2006). In SOA, the full logic of an application performing a task is decomposed into smaller, distinct units of logic that machines (i.e. servers) expose as services to other machines (i.e. clients) (Erl, 2005). The logic behind a service can be arbitrarily complex, ranging from full applications to simple actions (e.g., microservices). Due to services and interfaces heterogeneity, SOA needs ancillary services for finding and locating service producers (i.e., service registries) and to retrieve a description of the service interface (Salvendy and Karwowski, 2010).

Resource-oriented architectures are built around the concept of *resource*[2] that is *something supplying a want or deficiency*, or, as the World Wide Web Consortium (W3C) says, *an item of interest in the information space* (W3C, 2004). The rationale behind resource-orientation is that all pieces of information are available as resources through a uniform interface that requires a resource identifier. Since the interface must be the same for every kind of resource the allowed operations are necessarily low-level – typically mapping the CRUD pattern (Create-Retrieve-Update-Delete). Complex actions on resources are implemented through a concatenation of the basic operations.

The Web is the most well-known system based on a resource-oriented architecture (ROA). It was designed according to the Representational

[1] "from Latin servitium condition of a slave, body of slaves, from servus slave" [10, S.v. Service].

[2] "something that lies ready for use or that can be drawn upon for aid or to take care of a need" [14, S.v. Resource].

State Transfer (REST) architectural style including six constraints (Fielding, 2000), (Fielding and Taylor, 2002), (Fielding et al., 2017): Client-server, Stateless, Cache, Uniform interface, Layered system, and Code-on-demand. The Uniform Interface is the essential constraint of ROAs. It simply indicates that all the resources exposed by the system, must be accessible through the same interface. Therefore, no prior knowledge is necessary to interact with a resource. To better specify the Uniform Interface characteristics, the REST style specifies some sub-constraints: identification of resources, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine of application state.

Notably, the 'hypermedia as the engine of application state' constraint (sometimes shortened as HATEOAS) characterizes the REST style. Due to the Stateless constraint, REST stateful applications are designed as a state machine: the state of an application is stored in the representation of the currently accessed resource, including the hyperlinks that allow navigating towards other resources.

2.2.1. *Resource and service-orientation on the web*

The Web information space is built on the Internet and its implementation is based on a suite of specifications for its three architectural pillars (W3C, 2004).

- the Uniform Resource Identifier (URI) for Identification (Berners-Lee et al., 2005);
- the HyperText Transfer Protocol (HTTP) for Interaction (Nielsen et al., 1996); and
- the HyperText Markup Language (HTML) for Formats (Web Hypertext Application Technology Working Group (WHATWG)).

The three specifications URI-HTTP-HTML (with their extensions and descendants) are sufficient for implementing the Web space in full compliance with the REST architectural style. Yet, these specifications are quite flexible, providing a universal addressing schema (URI), a general-purpose request/response messaging protocol (HTTP), and an advanced hypertext format (HTML), which can be used beside and beyond their intended scope. This allowed developing services and applications using Web technologies but conforming to different architectural styles or at least not conforming with the original REST style.

2.2.2. *Web service-oriented architectures*

In the Web environment, building a service-oriented architecture (SOA) means dismissing most of its characteristics derived from the REST constraints. For example, the heterogeneity of interfaces makes the cache constraint mostly useless: rarely the same request will be repeated with an exact copy of the (many) parameters required by a typical service interface.

On the other hand, a service-oriented style introduces several additional architectural constraints to manage the provider/consumer interaction. Indeed, to implement the properly called Web Services, a full SOA protocol suite was defined, complementing URI and HTTP with other specifications – including SOAP for message transport, the Web Services Description Language (WSDL) for service description, the Universal Description Discovery and Integration (UDDI) for service registry, etc. (OASIS Open et al.).

2.2.3. *Web resource-oriented architectures*

On the Web, implementing a ROA, and specifically as a RESTful architecture, is easy since it is the style the Web is designed on. However, there are some subtleties in the REST style that should be considered to avoid breaking its constraints. Typical aspects to be considered are:

- *Not all the (new) Web technologies are compliant with the REST style*: for example, *cookies* –small chunks of information shared by clients and servers during the message exchanges – can easily violate the Stateless constraint (Barth, 2011). Indeed, due to cookies, two different users can have a completely different view of the same resource identified by its URI.
- *Application Programming Interfaces (APIs)*: It is common to read about "RESTful APIs" referring to parameter-based interfaces encoded in the URL, thus implementing some kind of Remote Procedure Call (RPC) over HTTP (RPC/HTTP). Instead, a real REST API is nothing more than a set of resources with an URI assigned and hypermedia representations. (Fielding) (Roy Fielding on Versioning, Hypermedia, and REST).

Of course, there is no specific obligation to be fully compliant with REST, but any violation should be the result of careful design to preserve the required features of the system.

The main limitation of the Web navigation paradigm is that a human user is needed to understand the content of a hypermedia document and then select the relevant hyperlink to trigger an application state transition. The concept of Semantic Web was proposed as an improvement of the Web to support machine-to-machine interaction, while keeping it conforming to the REST style (Berners-Lee et al., 2001). The Semantic Web concept consists in making the meaning of resources and hyperlinks explicit and machine-readable. The Semantic Web can be implemented by annotating resources and links by semantically enriching their metadata, or more generally, in a separated subgraph pointing to the resources. At the foundation of the Semantic Web lies the Resource Description Framework (RDF), a specification allowing to express statements about resources in the form of *subject–predicate–object*, known as triples (W3Cb). A triple can predicate a property about a resource (subject) with an assigned value (object) or it can predicate a relationship between two resources (subject and object). Subjects, predicates, and objects are all referred through URIs to make them readable. An agreement on shared vocabularies, thesauri and ontologies makes automated processing possible.

2.2.4. *Comparing resource and service-oriented architectures*

As expected, service and resource-oriented architectures differ in terms of capabilities, features and in terms of barriers for users and providers adoption.

- ❖ *Application design*
  - ➢ *SOAs are provider-driven*: providers decide which services to publish and hence which use cases to support. This allows providers to support arbitrarily complex use-cases, but it limits the possibility to enable new use-cases by new users, including system integrators.
  - ➢ *ROAs are user-driven*: providers expose the resources that they control, while users (e.g., application developers) decide how to combine them in a workflow that implements an application. This approach allows user to create new applications although it may be a complex task.
- ❖ *Infrastructure requirements*
  - ➢ SOAs are more demanding for providers: a) they must design and implement the interface to access their service; b) since services require execution, they must deploy them on a computing and storage infrastructure; c) they must estimate the required capabilities (number of requests, availability, security) to set up the service and possibly scale up.
  - ➢ ROAs have a lower entry barrier for providers. They only require exposing the resources, sharing their representations that can be, and often are, static. A common Web server is generally sufficient.

These differences make SOAs better fitting in environments where providers have strong IT expertise and capabilities, and use-cases are well-known – such as in domains like e-Government, e-Commerce, etc. On the other hand, ROAs fit well to open environments where resource providers may have little IT expertise or infrastructural capabilities, and use-cases are not defined in advance or can easily evolve. However, in this case the burden of application development is completely in the hand of a third party.

2.3. Mixed vs. multi-style architectures

Service-oriented and resource-oriented styles are, in principle, not compatible since the respective constraints conflict (e.g., dedicated interface vs. uniform interface). A mixed approach would disregard one or more constraints resulting in a null-style architecture and, thus, one with no characteristics guaranteed.

While mixed architectures are not doable, it is possible to have *multi-style architectures* where different subsystems, each based on its own style, communicate through dedicated *gateways*. Gateways are architectural components that lie at the boundaries of two subsystems and do not violate the respective constraints. For example, to make a RESTful and a SOA system interact, a gateway could expose a service through a resource such as a Web form to fill (Fig. 1), or a resource workflow as a service (Fig. 2).

2.4. Sharing of computational models

The *many* digital transformations of society provide, at the same time, both new opportunities and new challenges to Open Science. Digital infrastructures enable data and models sharing in the scientific communities (Chen et al., 2020). Focusing the attention to geospatial information – i.e., the information with an implicit or explicit reference to space and time – a big effort has been conducted in the last decades, to support data management and sharing. Despite lasting financial, technical, and legal barriers to data sharing, the so-called FAIR (Findability, Accessibility, Interoperability, and Reusability) data principles (Wilkinson et al., 2016) are now widely recognized by the scientific communities and recommended by research and innovation funding agencies in their programmes (European Commission Directorate-General for Research and Innovation, 2016), (European Commission, 2018). In parallel, technical solutions, including standard specifications, dedicated tools, brokered systems, cloud platforms, etc. make possible to implement operational systems for data sharing.

Concerning modelling, the situation is noticeably less mature (Laniak et al., 2013). Several efforts to facilitate model sharing, and possibly, interoperability and integration have been carried out in the last decades (Nativi et al., 2012). The simplest approach is the development of dedicated tools, like desktop applications, usually including a Graphical User Interface (GUI), to select a model and run it with proper parameters. Such approach, also termed as Model-as-a-Tool (MaaT), does not support a real interoperability and integration of models. Another proposed solution is the development of model frameworks, like the Open Modelling Interface (OpenMI), or model workflow engines (e.g., Taverna, Kepler) allowing to combine different models. However, they usually impose strict technological constraints on model developers and integrators, as requiring the adoption of a specific programming language or a development/deployment platform. These constraints act as a high barrier to interoperability - e.g., for the exploitation of legacy models. Moreover, they are often specialized for supporting specific community requirements, limiting multidisciplinary applications. More recently, these frameworks fully evolved towards Component-Based Architectures (CBAs), with heterogeneous model implementations dialoguing through well-defined interfaces, and later to service-oriented architectures for networked components.

2.4.1. Model sharing on the web

The idea of exposing models on global networked systems also enabling their combination to form loosely coupled integrated models has existed in various forms, for a long time. However, mostly due to technological constraints, it was not feasible until the Web become a mature platform for resources sharing. Around 2007, a vision of a "Model Web" was suggested by Gary Geller, Woody Turner and Forrest Melton for the ecological science domain (Geller and Turner, 2007), (Geller and Melton, 2008). They proposed the "Ecological Model Web"

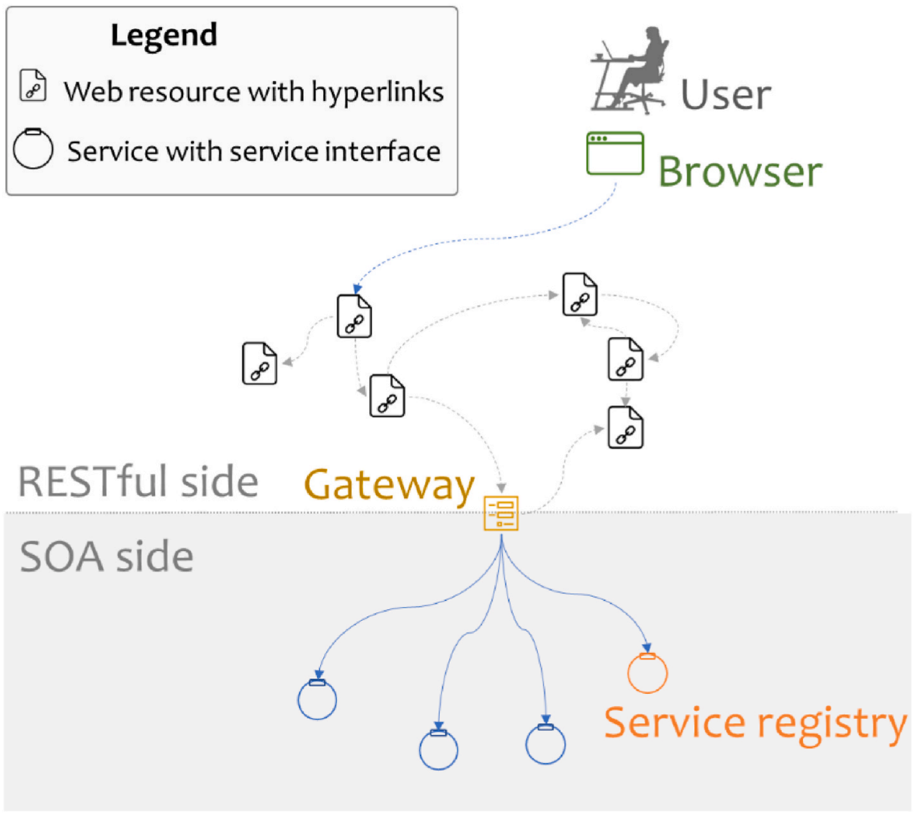

Fig. 1. Applying a multi-style architecture, a service is accessed through a resource (a Web form).

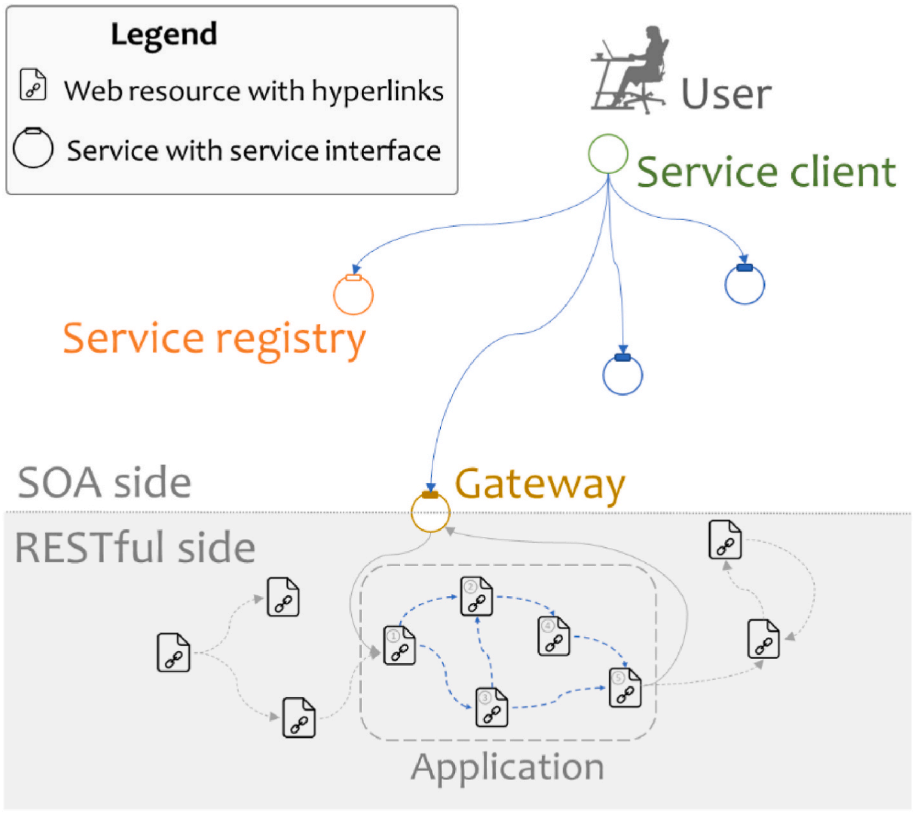

Fig. 2. Applying a multi-style architecture, a RESTful application is exposed as a service.

as "*a dynamic network of computer models that, together, can answer more questions than the individual models operating alone*". The authors clearly identified potential user scenarios (i.e., protected areas and natural ecosystems management) and challenges (i.e., models interoperability and stakeholders collaboration). In 2009, a "Model Web Development" task was initiated by the Group on Earth Observations (GEO) to develop "*a dynamic modelling infrastructure (Model Web) to serve researchers, managers, policy makers and the general public*" (Nativi et al., 2009a). The task activity (led by Gary Geller and Stefano Nativi) produced the definition of a Model Web Architecture consisting of: (i) a conceptual framework, (ii) a resource model, and (iii) a metadata framework (Nativi et al., 2009a). Contextually, some proofs-of-concept were elaborated for the natural ecosystem domain (Nativi et al., 2007), (Nativi et al., 2009b).

2.4.2. Model-sharing with service-oriented architectures: model-as-a service

Accessing a model usually means *running* it. Therefore, it is not surprising that the typical approach to sharing models is to offer the ability to run them remotely. This fits easily into the service-oriented approach, by defining a service called "run model". Providing online services for interacting with computational models is commonly referred to as the Model-as-a-Service (MaaS) approach.

In 2007, the Open Geospatial Consortium (OGC) released a first version of a Web Processing Service (WPS) specification for "*a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients*" (OGC, OpenGIS Web Processing Service). Although limited to geoprocessing, the scope was wide since a WPS "*may offer calculations as simple as subtracting one set of spatially referenced numbers from another […], or as complicated as a global climate change model*".

While WPS provides a general-purpose interface for processing any kind of vector or raster data, in 2008, OGC released a specification for a Web Coverage Processing Service interface focusing on coverage data (e.g., satellite imagery) (Baumann, 2010).

In 2015, OGC published a second version of WPS (WPS 2.0.2) (OGC) which adopted a different specification approach, defining "*a core conceptual model that may be used to specify a WPS in different architectures such as REST or SOAP*".

2.4.3. Model-sharing with resource-oriented architectures: model-as-a-resource

While the primary use of a computational model is to generate output through processing, providing a single service for running a model does not exhaust its potential. A computational model is a resource that can be exploited in several potential scenarios together with resources of other types. Users may be interested in understanding what the model does, how it was implemented, or deploying it on a high-performance computing (HPC) infrastructure for rapid processing.

The MaaS approach suffers from all the limitations of service-oriented systems. Overall, service providers define and limit resource usage (in this case, models). These providers decide which model to expose, which features are accessible remotely and which infrastructure to use (local machine or cluster, elastic cloud, etc.) thus limiting the non-functional aspects, e.g., input and output data size, processing time, etc.

It is interesting to evaluate the possibility to support interaction with computational models in a more open ROA.

In the past, some studies have been carried out to investigate and possibly demonstrate the feasibility of model sharing in a ROA. Most of these studies refer to ROAs in a general sense (Mazzetti et al., 2009), (Foerster et al., 2011), (52North), but a few also consider a potential RESTful implementation (Granell et al., 2013), (Flaishans et al., 2016). Nevertheless, the proposed solution is often just a translation of a 'run a model' service to a 'runnable model' resource, still missing many potentialities of a 'computational model' resource – for example, many implementations miss the point that the source code is the most relevant representation of a 'computational model' resource, for transparency and reusability. More recently, resource-oriented approaches have been explored: HydroShare for the hydrology community (Tarboton et al., 2024), and the GEO Infrastructure with its Knowledge Hub (Group on Earth Observations) in the Earth Observation domain, are two noticeable examples. The former specifically focuses on data and model sharing, while the latter focuses more generally on Open Knowledge support (GEO, 2021). However, the full exploitation of computational models, as digital resources, requires the definition of a new architectural framework that builds on successful experiences of both resource-oriented and service-oriented architectures.

3. System architecture for a MaaR framework

3.1. A model-as-a-resource framework to realize the Model Web vision

This section exposes the design of a distributed system architecture to share computational models as resources –i.e., a Model-as-a-Resource (MaaR) framework– and (in perspective) implement a Model Web. To this aim, a multi-style architecture (characterized by a RESTful core and a service-oriented subsystem for automating complex resource workflows) is proposed. This architectural solution provides the required functionalities to support high-level scenarios of Transparency, Reproducibility, Replicability, and Reusability, along with the relevant non-functional requirements –which are inheritably related to lowering entry barriers to providers and users of scientific models. To address (high-level) scenarios and use-cases, the main components of a MaaR framework are defined in the next sections. Being a complex system, MaaR components are specified by applying a view-based approach. Each viewpoint deals with the concerns of a stakeholder class. In keeping with the ISO Reference Model for Open Distributed Processing (RM-ODP) (ISO), the following viewpoints are considered: Enterprise, Computational, Information, Engineering, and Technology.

3.2. Enterprise viewpoint

The enterprise viewpoint "*is concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part*" (ISO). As such, scenarios and actors are the most important elements of this view.

3.2.1. Scenarios

The proposed MaaR framework aims at addressing four overarching Open Science scenarios.

- *Trasparency*: a user should be able to know what a computational model is, which scientific model it implements, which data it requires and produces, how it is implemented, etc.
- *Reproducibility*: a user should be able to reproduce a simulation experiment.
- *Replicability*: a user should be able to replicate the experiment in a different context (i.e., geographical area, temporal extent, data sources, etc.) Limitations on replicability can be defined in or derived by the computational model/experiment description.
- *Reusability*: a user should be able to reuse the simulation as part of a more general application. For example, a workflow for generating Land Cover Change maps could be used as part of a socio-economic model for decision-making. Again, limitations on reusability can be defined in or derived by the computational model/experiment description.

It is worth noting that the general Open Science scenarios above should not be considered (only) as scientific research scenarios. Transparency, Reproducibility, Replicability and Reusability are not only requirements for improving scientific knowledge, but also for improving accuracy of, and trust in science-informed decision-making (GEO, 2021). Replicability and Reusability are particularly important to enable the creation of knowledge products and services for addressing global changes that are typically multidisciplinary in nature.

3.2.2. Actors

Based on the previous description of scenarios we can identify some major actors.

- *End User*: End Users are the ultimate users of a product generated by the MaaR framework. For example, they may be decision-makers who make policy-relevant decisions based on a set of indicators generated with computational models processing EO data.
- *Intermediate User*: Intermediate Users directly interact with the MaaR framework. The most important Intermediate Users are the Application Providers, those who generate applications tailored to the End Users, using the functionalities offered by the MaaR framework.
- *Providers*: As the name implies, Providers provide the components of the MaaR framework. They include:
    - *Resource Providers*, who provide (and maintain) the information resources exposed by the MaaR, in particular datasets, computational models, knowledge artifacts (Data Provider, Model Provider, Knowledge Provider).
    - *Component Providers,* who provide (and maintain) the architectural components of the MaaR framework.
    - *Service Providers*, who provide (and maintain) services offered to other Users and Providers to facilitate their work. E.g., cloud service providers.

3.3. Information viewpoint

The information viewpoint "*is concerned with the kinds of information handled by the system and constraints on the use and interpretation of that information*" (ISO). It plays a fundamental role in a ROA since a resource is *an item of interest in the information space* (W3C, 2004) and its characterization as an information element is essential.

For the MaaR framework, the conceptual model that was proposed for the Model Web can be considered as a valuable starting point to identify the necessary resources (Fig. 3). It recognizes many resources, including the main abstract *Model* resource along with its potential representations (*ModelRepresentation*) and descriptions (*ModelMetadata*), differentiated by the multiple instances of *ModelRun* resources, and associated with input and output datasets.

3.3.1. Model resources

3.3.1.1. Models as algorithms. To build the Model Web by applying a MaaR approach, the most important resource is the Model object. As introduced in the previous sections, in a MaaR framework, a Model resource refers to a scientific model, with its computational representation as an optional part of its description –although required for running the model on a computer system.

But what a Model resource, exactly, is? There are many potential answers. The broadest one is that a Model is identified by how it relates input and output values (also referred as input/output semantic equivalence) (Lastovetsky and Gaissaryan, 1994). The strictest one is that a Model is identified by the program that effectively computes the output value from the input value. However, none of them captures the intuitive notion of a Model resource: the former only considers the observable behavior (input/output) losing any reference to how the Model represents the reality; the latter does not distinguish between relevant and irrelevant differences in implementing programs.

Computer science distinguishes among computation results, algorithms and programs, based on the logic concepts of syntax vs. semantics, and sense vs. denotation. According to Moschovakis (1993), programs are syntactic objects, while "*algorithms are semantic (mathematical, set theoretic) objects*" explaining how to generate an output. The algorithm must be considered the *sense* of a program while its output value is its *denotation* - "*A program is a piece of text, it means nothing uninterpreted; its interpretation (or one of them) is precisely the algorithm it*

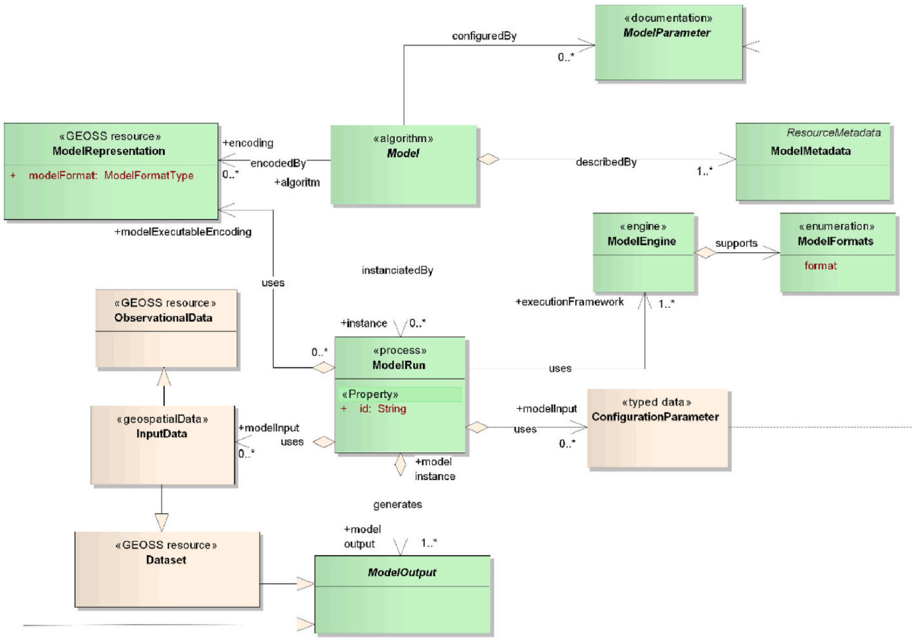

Fig. 3. The Model Web data model – from (Nativi et al., 2012).

defines, and that algorithm is no longer a syntactic object". This definition of an algorithm provides an intermediate concept between the observable input/output relationship and the implementing program, which fits well to the concept of Model resource. Therefore, in the proposed MaaR framework, *a Model resource is the algorithm that computationally describes a scientific representational model*. In such a way, a strong semantic equivalence is adopted, avoiding both the weak semantic equivalence and the syntactic equivalence: a) two Models computing the same output for the same input (weak equivalence) are not necessarily equivalent, and b) two Models realized by different programs (syntactic difference) are not necessarily different.

3.3.2. Other model resources

By identifying models with algorithms, it is possible to assume that models can have multiple realizations encoded as syntactically different programs. Software programs are considered just as possible representations of the same Model resource –not having to introduce further resources in the schema. However, this choice would have some drawbacks, because in general a program (as a source code) does not include all the necessary information –e.g., libraries, software framework versions – for its execution, which is one of the main objectives of a MaaR framework. To address that, a dedicated *Model Implementation* resource is introduced, which logically collects all the necessary information to run a Model. In addition, a Model may have one or more *Model Description* resources, which explain the scientific background of the representational model. Finally, a model execution (i.e., a run) establishes a relationship between specific input and output datasets, as well as potential further conditions. Therefore, it is convenient to create dedicated *Model Run* resources to store this kind of information.

In summary, the core MaaR resources are.

- *Model* resource: the algorithm that describes a scientific representational model.
- *Model Implementation* resource: the set of information required to run a scientific representational model, including implementations as computer programs.
- *Model Description* resource: a piece of information about the scientific background of a representational model.
- *Model Run* resource: the context of a specific model run.

It is expected that a full implementation of a MaaR framework will introduce further ancillary resources that are accessible to different sets of users - e.g., model inventory, model run inventories.

3.3.3. Data resources

For a MaaR framework, the second important category of resources is data. Dealing with data is much easier than with models because data are commonly managed as statical resources; moreover, there is a lot of conceptualization and standardization work (carried out in the last decades) to leverage on data description and representations at global and community level (ISO/TC 211), (OGC Standards), (Sansone et al., 2019). Relationship between data and models is largely missing - e.g., the possibility to formalize that a particular dataset is an input, or an output of a given model.

For the MaaR framework, the Data resources taxonomy includes.

- *Data* resource: the set of values of parameters or indices.
- *Data Description* resource: a piece of information about the background of data - e.g., accuracy, how they have been obtained.

3.3.4. MaaR framework main resources

The main resources and their relationships characterizing a MaaR framework are depicted in Fig. 4. The conceptual schema defines that.

1. The *Model Description* resources should provide the constraints on the *Model* use, including for example the types of input and output data, the geographical coverage, the temporal extents, and all the assumptions to correctly apply the model.
2. The *Model Implementation* resource should include the *Model Code*, which is the program implementing the model algorithm as well as all the necessary information to execute it –such as the specificities of the running environment.
3. The *Model Run* resource is associated with *Input* and *Output Data* that, in turn, should be compliant with (i.e., should realize) the respective constraints.

Noteworthy, the potential parametrization of the model can be expressed as part of the model description. However, since a parametrized model can be considered as a collection of models that are indexed by parameters values, the parameters can be logically considered as

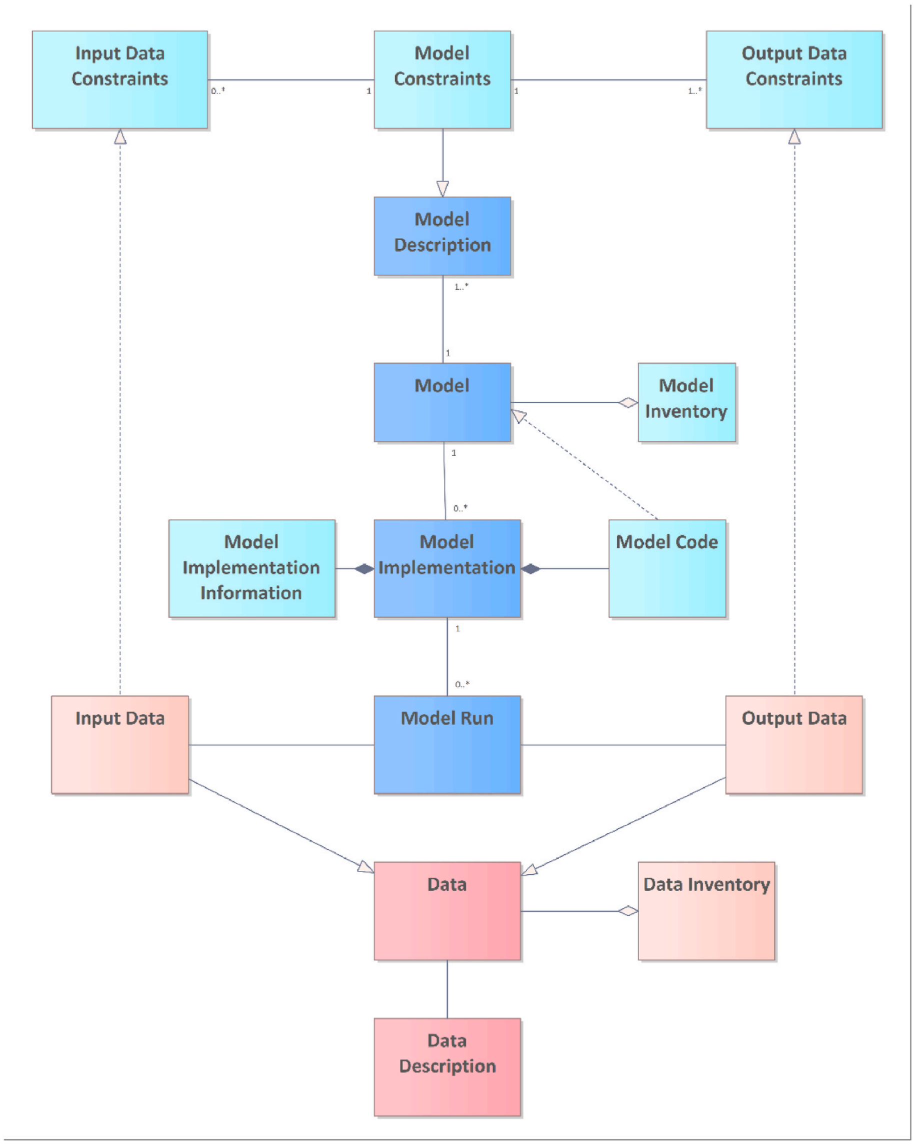

Fig. 4. Main resources of a MaaR framework.

Input Data –see the Kleene's Parametrization Theorem (Odifreddi, 1989).

3.4. Computational viewpoint

The Computational viewpoint "*is concerned with the functional decomposition of the system into a set of objects that interact at interfaces - enabling system distribution*" (ISO).

The adoption of a resource-oriented architectural style greatly simplifies the computational viewpoint. ROAs move the system complexity from the operations definition to the resources identification and description making the functional decomposition relatively easy – by leveraging the Uniform Interface constraint. Mapping the CRUD pattern, the system must be able to support high-level functionalities of resource management (to create, update and delete resources) and access – to retrieve and present resources.

3.4.1. Building applications: hypermedia as the engine of application state

To develop specific applications, the REST style builds on the use of hypermedia: an application is created as a workflow of interactions with resources. A couple of examples of use-cases can show how a RESTful application can be built based on the proposed resources and some ancillary resources that are necessary for specific cases.

3.4.1.1. A transparency use-case. **Description**: "*A user finds a model M and the information that, through the model, it is possible to generate a dataset DO by ingesting a dataset DI. The user collects information about the model.*" (see Table 1)

To implement this use-case the user needs an entry point expressed as a URI. The entry-point may be a Web form to search a catalogue/digital library of models (*Model Inventory* resource) based on an inventory of models with filtering functionalities accessible through the query part of the URI.

3.4.1.2. Reproducibility, replicability use-case. **Description**: "*A user finds a model M and the information that the model M can generate a dataset DO by ingesting a dataset DI. The user run the model to reproduce the result cited in the model description and to replicate it on a different scenario*". (see Table 2)

HATEOAS constraint induces to design applications as state machines, where the hypermedia documents represent a state, and the hyperlinks activate the state transitions. Fig. 5 shows the Reproducibility and Replicability use case represented as a state machine.

The two examples, previously discussed, show the flexibility of MaaR

Table 1
Transparency use case description.

Step	User	System
1	The user accesses the *Model Inventory* resource.	The system answers presenting a Model Catalog form with fields for filtered search.
2	The user fills in the fields and launches the query.	The system answers presenting a form with fields for filtered search and the results of the previous query as a list of links to *Model* resources with a short description.
3	The user selects the link of interest.	The system answers presenting the basic representation of the selected *Model* resource with a backlink to the *Model Inventory* resource, links to *Model Description* resources, links to *Model Implementation* resources and links to previously generated *Model Run* resources.
4	The user selects one *Model Description* resource.	The system answers with a scientific paper informing that the model produced a significant scientific result.
5	The user goes back and selects another *Model Description* resource.	The system answers with a scientific paper describing the scientific basis of the model.

Table 2
Reproducibility and Replicability use cases description.

Step	User	System
1.4	*(As in the Transparency use-case)*	*(As in the Transparency use-case)*
5	The user goes back and selects one *Model Implementation* resource.	The system answers with information about an implementation of the model in Python programming language and links to a Git repository containing the source code (*Model Code*), and to a *Model Form* resource for execution.
6	The user goes to the *Model Form* resource.	The system answers with a form including a map for selecting a geographical area, a calendar for selecting a temporal extent, and a drop-down menu for selecting the input datasets. The form has also a drop-down menu of predefined scenarios.
7	The user selects one of the predefined scenarios (Reproducibility) cited in the *Model Description* previously read, and start the model.	The system answers with some information and a link to the generated dataset.
8	The user downloads the generated datasets and locally verifies that it corresponds to what the *Model Description* says.	
9	The user goes back to the *Model Form* resource.	The system answers with a form including a map for selecting a geographical area, a calendar for selecting a temporal extent, and a drop-down menu for selecting the input datasets. The form has also a drop-down menu of predefined scenarios.
10	The user defines a new scenario (Replicability) selecting a new location, time and/or input data and runs the model.	The system answers with some information and a link to the generated dataset.
11	The user downloads the generated datasets.	

and RESTful approaches. Through the simple definition of a core set of resources (along with a careful design of their representations) and by adding some ancillary resources (e.g., dedicated forms), it is possible to support complex use-cases implementation – via the introduction and encoding of the different application states. The examples show some limitations and drawbacks, too.

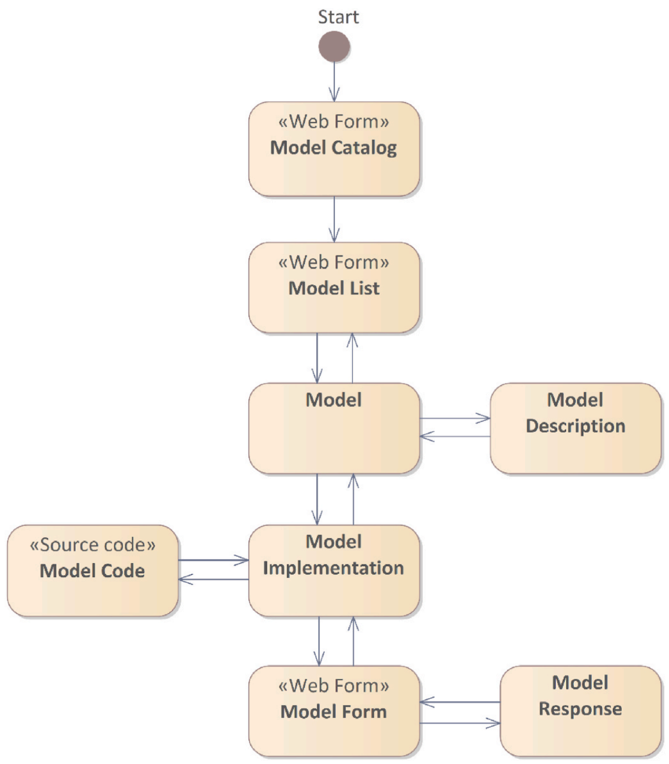

Fig. 5. The Reproducibility and Replicability application as a state machine.

- For Reproducibility and Replicability, the model run is strongly constrained by the provider's capabilities. The user cannot do more than what is proposed by the Model Form.
- A full Reusability use-case, where a model becomes part of a more general workflow, is rather complex to implement.

However, it is worth noting that resource-orientation enables also offline completion of similar use-cases.

3.4.1.3. Reproducibility, replicability use-case (offline alternative). **Description**: "*A user finds a model M and the information that, through the model, it is possible to generate a dataset DO by ingesting a dataset DI. The user run the model to reproduce the result cited in the model description and to replicate it on a different scenario*." (see Table 3)

Although this may be a non-optimal solution, it is something that service-oriented architectures cannot provide. With service-orientation, users have no alternative but utilizing the proposed MaaS service. Information for building and running a model is part of the description of a

Table 3
Reproducibility, Replicability use-case (offline alternative) description.

Step	User	System
1.4	*(As in the Transparency use-case)*	*(As in the Transparency use-case)*
5	The user goes back and selects one *Model Implementation* resource.	The system answers with information about an implementation of the model in Python programming language and links to a Git repository containing the source code, and to a *Model Form* resource for execution.
6	The user selects the Git link.	The system directs the user to the Git project landing page.
7	The user builds the model, prepares a Docker container, moves it on a cloud platform, and runs it.	

model resource, but it is not properly part of the information about a modelling service. In a service-oriented architecture, the provision of the information useful for building and running a model on a given infrastructure makes no sense, instead it perfectly fits with a ROA.

3.4.2. Service integration

The offline alternative use-case hides a major issue: the user is required to do highly heterogeneous actions. While the online use-case perfectly fits to a scientist, or a domain expert supporting a decision-maker, the offline use-case, at step #7, requires that the user "builds the model, prepares a Docker container, moves it on a cloud platform, and runs it". These actions are commonly beyond the competence of "a scientist, or a domain expert supporting a decision-maker". It is possible to redefine the "user" meaning, e.g., referring to a team of people with complementary expertise. Alternatively, there is the need to automate (as much as possible) the tasks that are outside scientist's expertise. Interestingly, this is the typical situation where a dedicated "service" is needed –i.e., the scientist needs a service to run the model. The service can be offered by humans (e.g., a software expert, or a cloud administrator) or by machines. The second approach requires a service-oriented architecture and that is the reason why a multi-style architecture better meets the MaaR framework requirements. In general, every complex action, which can be effectively automated, might be offered as a dedicated service. To keep the advantages of a ROA, those services should be offered through proper gateways that generate resource representations from services.

This example proposes some of the functionalities that can greatly improve a MaaR framework; they can be offered as services hidden behind resource gateways. Fig. 6 shows the main components for an enhanced MaaR framework that support the navigation paradigm for user interaction, and advanced services for model access and execution.

Table 4 describes the components depicted in Fig. 6.

3.4.3. Synchronous vs asynchronous interaction patterns

The examples above suggest the adoption of a synchronous interaction pattern with the user sending a request and waiting for the response

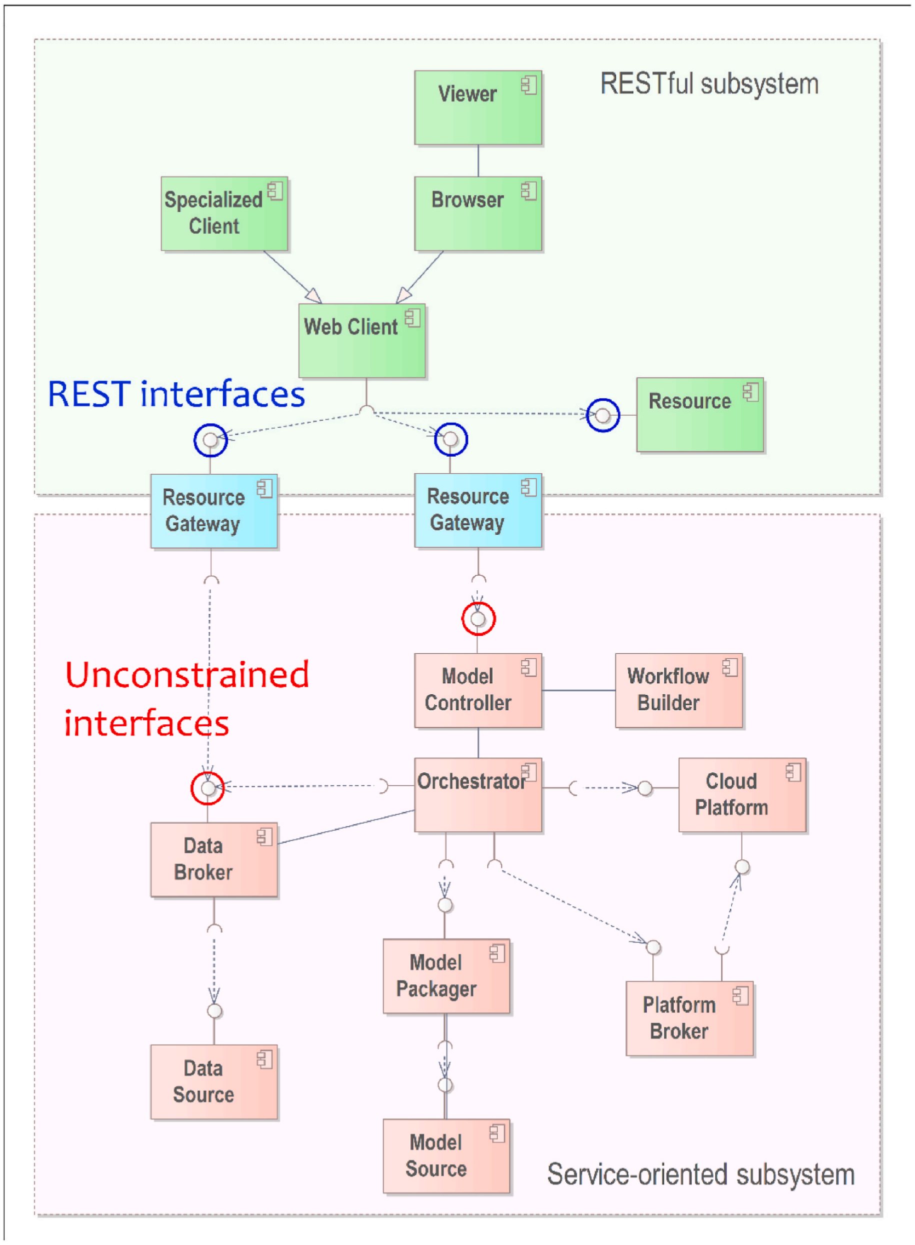

Fig. 6. Main components of an enhanced MaaR framework and relevant interfaces.

Table 4
Main components of a MaaR framework.

Component	Description
Web Client	A Web Client provides the interaction with resources through calls to the Uniform Interface.
Browser	The Browser is the typical Web Client.
Viewer	The Viewer is a component associated to a Browser and providing the presentation of content (resources representations)
Specialized Client	A Specialized Client is a Web Client tailored to specific usage scenarios.
Resource	A Resource is any kind of information accessible through a Uniform Interface implementation
Resource Gateway	A Resource Gateway is a façade component that exposes a Uniform Interface, but it can interact with services in a service-oriented environment. Fig. 6 shows two examples of Resource Gateways: the first one exposes data that are accessible through a data service, as resources; the second one allows running a model as a resource calling the Model Controller.
Model Controller	The Model Controller provides a service for running a model. It associates the request to a workflow of data access and model invocation and passes it to an Orchestrator for the execution.
Workflow Builder	The Workflow Builder provides workflows associated to the request managed by the Model Controller
Orchestrator	The Orchestrator is the core component for model execution. It coordinates the invocation of services needed to execute the model run requested by a user.
Data Broker	A Data Broker is a component that interacts with Data Sources accessible as data services. It implements basic transformations (change of format, reprojection, resampling, etc.) that facilitate the model execution.
Data Source	A Data Source provides datasets exposing a dedicated service interface. A Model Packager is able to recreate the software environment required to run a model and provide it as a self-contained package (e.g. container, virtual machine) for execution on an external Cloud Platform.
Model Packager	A Model Packager is able to recreate the software environment required to run a model and provide it as a self-contained package (e.g. container, virtual machine) for execution on an external Cloud Platform.
Model Source	A Model Source provides the model source code exposing a dedicated service interface.
Platform Broker	A Platform Broker provides a matchmaking service between user needs and Cloud Platform offering.
Cloud Platform	A Cloud Platform provides a set of services for model run (e.g., storage, elastic computing, container orchestration, etc.).

("The system answers with some information and a link to the generated dataset."). For an infrastructure supporting model execution that could have long run time and potential failures, an asynchronous interaction pattern is likely necessary. Typically, this can be implemented by generating an immediate response with the URL where the user can access the resource, once it will be ready. More advanced implementations can make use of a 'publish and subscribe' interaction pattern.

3.4.4. Real-time and near-real-time interaction

Another important question is whether and how the proposed framework is able to cope with the requirements of real-time (RT) and near-real-time (NRT) communication that many computational models have. Of course, if the RT/NRT requirement does not affect the Web resource sharing – i.e., it is hidden in the service-oriented subsystem – then it can be easily solved with dedicated interfaces. Instead, it may seem a challenge if the RT/NRT communication must be supported by the REST interfaces (e.g., for presenting real-time data to a user) since HTTP as the most common Web protocol is a Request/Response protocol. However, it is worth noting that the REST architectural style does not impose any specific communication protocol or format since they can be both negotiated as part of the interface exchange. Referring to Fig. 6, an example of interaction at the REST interface – e.g., already supported by modern browsers - may be.

1. A logical resource with real-time content (e.g., "current temperature at location X″) is identified by a URL, e.g., http://www/example.org/xyz/42 which informs about the resource location.
2. The Web Client accesses the resource using the HTTP protocol since the URL specifies 'http' as its 'scheme' portion (protocol negotiation) (Berners-Lee et al., 2005).
3. Through the Resource Gateway, the Web Client receives a Web page as the preferred resource representation. The response informs that the content is HTML with the Content-Type header (format negotiation). Then, the Web Client passes the HTML content to its internal HTML viewer component.
4. The HTML page includes a JavaScript code identified by a '<SCRIPT src = "text/javascript">' tag. Since code-on-demand is a REST constraint, running client-side code is REST compliant. The Web Client passes the code to its internal Javascript engine.
5. The Javascript code uses Websocket API to open a two-way communication with the real-time data service (Melnikov and Fette, 2011). Through Javascript the real-time data are presented to the user as dynamic representation.

Note that this solution does not violate any REST constraints. In particular, the interaction still happen at the Uniform Interface with a logical resource identified by a URI and the interaction is Stateless, since the representation does not depend on the previous client's requests.

This is just an example of a possible REST-compliant solution using available technologies. Many solutions can be adopted to support RT/NRT communication, as far as the REST constraints, in particular the Uniform Interface one, is respected. Other solutions could be based on resource polling if there is not a strict RT requirement, or on the use of different schemes (protocols) with dedicated clients.

3.4.5. Knowledge assets

The introduced resource categories enable general MaaR scenarios. More sophisticated use cases may require further information concerning the resources and their contextual role. These relationships, which can be inferred by a human actor from some text description (e.g., the model constraints), must be fully formalized for machine-to-machine interactions. For instance, by reading a paper, a scientist can discover that a model needs a 'precipitation' data input, and it was tested on the Mediterranean geographic area. For a machine client, this information needs to be formalized according to a Semantic Web approach - i.e., building a graph of resources (Mazzetti et al., 2022).

3.5. Engineering viewpoint

The Engineering viewpoint "*is concerned with the infrastructure required to support system distribution*" (ISO). A MaaR framework can be implemented as a three-tier architecture consisting of:

- **Client tier**: collecting all the nodes accessing resources and interacting with them.
- **Application tier**: collecting all the nodes providing services needed for building applications.
- **Resource tier**: collecting all the nodes that provide data and computational resources.

Fig. 7 shows an example of deployment, assigning the relevant functionalities to dedicated engineering components deployed on different nodes. The schema depicts Clients (gathered in the Client tier), Servers (offering access to data and model services gathered in the Application tier), and finally the resource provision servers –collected in the Resource tier. In this case a Client (PC) is equipped with Browser and Viewer components to interact with Web Servers and Web Application Servers. A Web Application Server accesses a Data Broker, which in turns accesses Data Sources provided as services. Another Web Application Server accesses the Model run functionalities, which are offered

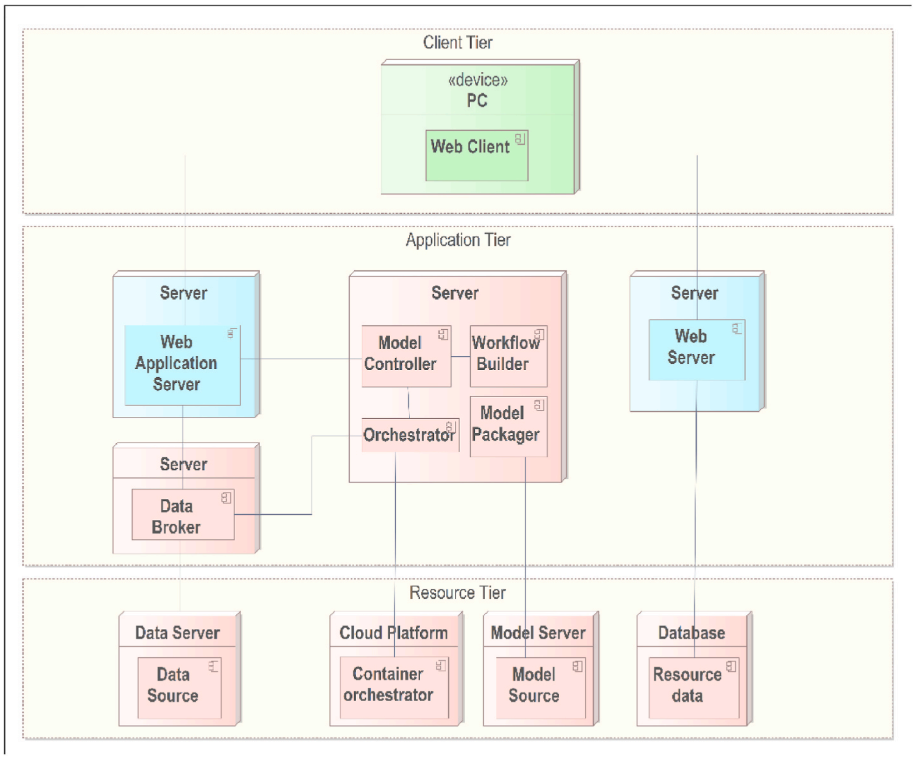

Fig. 7. Example of components deployment.

by a Model Controller, a Workflow Builder, an Orchestrator, and a Model Packager. All these components are hosted on the same node for tight coupling and/or better performances. A Web Server accesses databases that manage (static) resources such as Web pages, model code, and structured data. According to the navigation paradigm, Clients interact only with Web Application Servers and Web Servers –which act as the gateways between the RESTful architecture and the service-oriented architecture.

3.6. *Technology viewpoint*

The Technology viewpoint "*is concerned with the choice of technology to support system distribution*" (ISO). The proposed MaaR framework is based on a multi-style architecture including a RESTful subsystem as well as a service-oriented subsystem, which offers advanced functionalities to generate resources representations. Such solution can be implemented by leveraging several mature and innovative technologies that address the different aspects and provide the necessary capabilities.

3.6.1. *Communication technologies*

Fig. 7 shows a possible technological deployment to implement a MaaR framework. The deployment includes nine different types of nodes that provide different capabilities and interact for enabling different MaaR use-cases. Interoperability at different levels is a major requirement. Referring to the Level of Conceptual Interoperability Model (LCIM) (Tolk et al., 2007), it is possible to assume that the TCP/IP suite provides Technical Interoperability (Level 1). In the RESTful subsystem, Syntactic Interoperability (Level 2) can be achieved through the two core Web protocols (URI, HTTP) for Identification (of resources) and Interaction (with resources) respectively. In a MaaR framework, the characteristic of exchanged resources (e.g., data and models) makes necessary to support specific representation formats. Beside common hypermedia and multimedia formats (e.g., XHTML, JPG, PNG, MP4, etc.) clients should be able to present formats specifically defined for representing models and geospatial data. The Code-on-Demand constraint of the REST style allows enhancing client capabilities to include data viewers (in or out of the browser) supporting the most common formats like: GeoTiff, NetCDF, HDF, etc. Concerning data and model descriptions, the work done by the major standardization bodies in the geospatial domain (like ISO and OGC) provides a strong ground for client/server interoperability. The Code-on-Demand constraint can also help to overcome HTTP limitations, e.g. to better support asynchronous interactions and real-time/near-real-time communication.

We can safely assume that URI (for service addressing) and HTTP (as transport protocol for service payload) are also used in the service-oriented subsystem. However, they are not sufficient and must be complemented with service interface specifications. As anticipated in the "Web service-oriented architectures" section, on the Web, there is no agreement on the adoption of a unique service-oriented stack. Some services are offered according to the SOAP suite, others through proprietary specifications or light RPC-like APIs. This is also the situation for geospatial data and model sharing, where services are mostly exposed according to OGC specifications or by implementing proprietary APIs.

Semantic Interoperability (Level 3) is not strictly necessary for a MaaR framework. However, the availability of semantic information would greatly improve MaaR use-cases. RDF, RDF schema, OWL, etc. are mature specifications for knowledge encoding. In addition, many knowledge bodies (i.e., ontologies, taxonomies, thesauri, vocabularies, conventions) are available to express the semantics of geospatial, environmental, and scientific domains. Many initiatives on standard specifications have defined semantics aspects as part of their activities on the description of data and services (Villa et al., 2017). However, a big effort is still necessary to align and harmonize the existing knowledge bodies. On the technological side, several RDF triple stores are available, both as commercial and open-source tools, to store graphs representing domain knowledge. The major obstacle in semantic interoperability does not seem to be the lack of technology, but a governance issue: how to define an effective process to collect, formalize, and encode the experts' knowledge.

Pragmatic Interoperability (level 4) faces the same challenges of semantic interoperability, but at a higher level. Pragmatic interoperability deals with the possible use of data and models in a specific context. This can be achieved by enriching the data and model description with information about data quality and fitness-for-purpose. In the last decades, several initiatives have been launched on the description of geospatial data quality. However, there is not a general agreement on the subset of information that is needed to qualify a dataset or a model, as useable in a specific scientific context. Minimal information should include spatial and temporal coverage and resolution, as well as accuracy. QualityML (Quality Indicators Dictionary and Markup Language) is a profile of the ISO geospatial metadata standards (e.g., ISO, 19157) providing a set of rules for precisely documenting quality measure parameters. It includes semantics and vocabularies for the quality concepts (Ninyerola et al., 2014), (QualityML).

A specific mention must be dedicated to uncertainty, which is a fundamental information for many application use-cases, when geospatial data and models are used in support of environmental decision-making. Data and model description should provide information on the uncertainty associated with data and how it propagates in models to make possible estimating the uncertainty of a workflow output – a piece of information that is a necessary for any effective decision. Uncertainty description can be achieved by combining related metadata and external annotations. UncertML (Uncertainty Markup Language) is "*an XML schema for describing uncertain information, which is capable of describing a range of uncertain quantities*" proposed as OGC Discussion paper (OGC, 2009). It defines a general conceptual model, that allows uncertainty to be quantified in a variety of ways –i.e., realizations, statistics, and probability distributions. UncertML has been experimented in different contexts according to the Linked Data and Semantic Web approach (Williams et al., 2008), in integration with sensors (Stasch et al., 2012), with quality (Ninyerola et al., 2014), (QualityML) and provenance information (Car et al., 2015) and, more specifically, for the Model Web (Bastin et al., 2013).

3.6.2. *Mediation technologies*

The proposed MaaR framework is intended to lower present barriers to both users and providers. This is pursued by keeping interoperability agreements at the minimum and avoiding making any assumption on how services are offered. As a consequence, the service-oriented subsystem must be considered (in principle) highly heterogeneous (Nativi et al., 2004): data sources can publish data via many different protocols, ranging from complex and powerful interfaces (such as WCPS) to light and simple interfaces like shared folders; models can be published as web services, as containers or virtual-machines, they can be even offered as open-source code – along with the instructions to compile and run them. Finally, data and models can be described and annotated according to heterogeneous ontologies or knowledge bodies.

A solution to address heterogeneity without imposing heavy constraints to providers and/or users is the adoption of a mediation pattern and the introduction of dedicated components that harmonize the existing services (Nativi et al., 2013). A couple of functional components (the Data Broker and the Model brokering framework, including Model Controller, Workflow Builder, Model Packager and Orchestrator) were introduced specifically to offer mediation and harmonization services at syntactic level (Fig. 6).

The Data Broker should dialogue with servers, by interacting through (international and community) open standards and widespread protocols for geospatial data discovery and access (such as OGC WxS, WPS, OpenSearch, CKAN, etc.), but also with legacy systems, which make use of proprietary protocols. On the other hand, the broker must expose standard common interfaces to clients. Several data mediators already exist. They are often part of more general data storage and processing systems. For example, data cube or business intelligence solutions typically include components dedicated to extract, transform and load data (ETL process) for ingestion. However, they are tailored for their target system and cannot be easily adapted to wider scenarios. In the geospatial world, there are also stand-alone data mediation solutions. For instance, NOAA ERDDAP provides a virtual data server offering scientific data transformation capabilities through widespread interfaces (i.e., OpenDAP, RESTful APIs) (NOAA). Likely, the most comprehensive stand-alone solution is the Discovery and Access Broker (DAB), adopted by GEO (Group of Earth Observation) for its Global Earth Observation System of Systems (GEOSS) and by WMO for its WMO Hydrological Observing System (WHOS) (Boldrini et al., 2022). The DAB supports a great number of specifications and profiles for geospatial data discovery and access, and it is deployable on scalable cloud infrastructures for high performances (CNR).

The Model broker is a more complicated component that should interact with existing processing platforms, offered through heterogeneous service interfaces and APIs. In addition, it must create the processing services for those models that are provided as source code. In this case, the broker must build the software environment requested by the model, and then deploy it to make it accessible as a service. Virtualization techniques provide mature technologies for creating a self-contained virtual machine that can be hosted on different systems. More recently, containerization technologies provided a lighter solution than virtualization (Watada et al., 2019). In the last years, the Docker technology became the de-facto standard for containerization. Most commercial and open hosting providers and cloud platforms offer the possibility to run Docker container images, and to configure and orchestrate them with advanced services (e.g., Kubernetes). Assuming that a Docker configuration file is provided along with the model source code, a Model brokering framework could use it for compiling the model, build the necessary Docker container image and deploy it for execution on any available platform. For example, this is the approach adopted by the Virtual Earth Laboratory (VLab) for experimenting the Model Web implementation for data to knowledge use cases (Santoro et al., 2016), (Santoro et al., 2020).

The Data and Model brokers address syntactic interoperability. They do not make any use of semantic information. The Data Broker transforms geospatial metadata and data without any reference to their content; the Model brokering framework builds and run models independently of what they do. In principle, a mediation and brokering approach could be adopted for pursuing the semantic and pragmatic interoperability. Instead of imposing a common ontology to the knowledge providers, a dedicated Knowledge Broker might map different ontologies on a metamodel. However, for the time being, the semantic description of dataset and models is still limited, and it is not possible to evaluate the validity of existing knowledge mediation technologies.

3.6.3. *At the boundary: the APIs role*

Commonly, different users interact with the MaaR framework in different ways. The description above focuses on a couple of use-cases – i.e., transparency and replicability/reusability. It is useful to add something about the interaction of other user categories with the framework.

For resources providers, to be part of the system is made easy by the openness of the MaaR framework. Providers can simply publish their resources offering a basic representation of them through a Web Server. For example, a model provider could make a model accessible as a source code on a Web accessible Git repository and a data provider could publish datasets through an existing Web Service. However, the full integration of a resource in the MaaR framework requires some minimal interoperability agreements. For example, a Git repository does not require providing information about containerization (e.g., a Docker configuration file). Still, the system openness and extensibility make possible to integrate missing information, e.g., with a third-party adding containerization information through a different Web Server.

Service providers are key actors in the life cycle of the framework since they add value by supplying services that automate complex procedures. For example, a service provider could design, implement, and maintain a Model brokering service, which is able to collect source code and containerization instructions (from disparate sources) and automate the compilation and running of models. Other providers might offer knowledge services that enable semantic queries on harvested metadata.

Application developers build the applications, in the MaaR framework. According to the REST architectural style, applications must be designed as resource state machines, but they can use the services available in the backend service-oriented subsystem.

In the last years, to facilitate Web application development, the concept of API (Application Programming Interface) has gained increasing attention. They are interfaces specifically implemented to assist application developers.

In the context of the MaaR framework, it is important to distinguish between REST interfaces and general unconstrained APIs and where they are intended to be used in the proposed architecture. Referring to the logical component diagram of the MaaR framework (Fig. 6), REST interfaces must be exposed to MaaR framework consumers (e.g., browsers), while unconstrained APIs can be offered by those tools (like a Model Controller or a Data Broker instance) that are useful to build MaaR applications. Other interfaces might already be part of a service-oriented subsystem with its own interoperability agreements. Fig. 6 highlights the role of Resource Gateways that are explicitly introduced to expose a REST interface that enables the proper Web interaction.

4. Scientific and technical contribution

4.1. The MaaR framework as the basis for a digital ecosystem

The openness of the proposed architecture facilitates the participation of different users with different expertise, suggesting it as the core framework for a potential digital ecosystem. Nativi and Craglia (2021), and Nativi, Mazzetti and Craglia (Nativi et al., 2021) discuss how digital ecosystems can be used to realize geospatial digital twins, identifying challenges and opportunities; Annoni et al. (2023) propose the digital ecosystem approach as one of the basis for the Digital Earth concept. The MaaR framework is aligned with that vision and presents important benefits in such a direction. Building a digital ecosystem around a MaaR framework would allow starting a coevolution process that can enrich the ecosystem with new resources, and in turn enable more and more knowledge-generation processes –new users and providers can join the ecosystem and provide new services and resources for knowledge generation (e.g. resources annotation services to address semantic interoperability, workflow schemes to build applications and to enable models interoperability, brokers to face cloud infrastructures and platforms interoperability).

Applying a multi-style approach has also a positive impact in terms of governance because it adopts a clear separation-of-concern pattern. Each actor can focus on his/her specific expertise, while the open architecture allows new actors to enter on the stage. For example, a knowledge provider could implement a graph database that experts can fill in providing knowledge about existing resources –e.g., datasets and scientific models. Therefore, the MaaR framework is a good candidate to build a digital ecosystem with different "digital species" collaborating and competing on the same digital environment and contributing to the overall ecosystem service of knowledge generation (Nativi et al., 2021). To face internal and external changes and evolutions, the ecosystem governance must assure the invariance of few essential traits. The most important invariant of the proposed MaaR framework is its multi-style architecture –see the computational view of Fig. 6. It is important that any system evolution (at the enterprise, information, computational, engineering, and technological level) keep this trait unchanged, because several advantages depend on the separation of the RESTful and service-oriented subsystems. Without this separation, the architecture would collapse into a mixed architecture (i.e., at most a layered architecture) with very few characteristics guaranteed.

4.2. Comparison with other solutions

The increasing importance of computational modelling in the scientific practice, and the pervasiveness of the Web, has meant that technological solutions have been proposed to support the sharing of computational models in the Web. In some cases, full infrastructures for data and model sharing have been implemented making use of widespread Web technologies. Two noticeable examples are HydroShare in the hydrology community, and the GEO Infrastructure with its Knowledge Hub in the Earth Observation domain.

HydroShare: HydroShare is an open source, web-based hydrologic information system developed for researchers, scientists, and data managers in the hydrologic sciences to easily share and publish data, models, scripts, and applications associated with research projects and resulting manuscripts (Tarboton et al., 2024), (Essawy et al., 2018). From the Model Web point-of-view, HydroShare is a valuable operational example of how, adopting a ROA, it is possible to share models as resources and how current Web technologies can support it. As such, HydroShare could gain benefits from the MaaR design that we present, as a conceptual framework that helps to assure the viability of evolving Web-based systems.

GEO Knowledge Hub: The GEO Knowledge Hub is an open-source digital repository of open, authoritative, and reproducible knowledge created by the Group on Earth Observations (GEO). In the GEO Knowledge Hub, EO Applications are organized in Knowledge Packages (Group on Earth Observations), (Carlos et al., 2022). Although the support for Open Knowledge is the clear objective, the GEO Knowledge Hub is currently a digital repository with limited support for model sharing. The GEO Knowledge Hub and the GEOSS Platform for data sharing could benefit from the proposed architectural framework to evolve towards a Model Web implementation.

In recent years, some technologies have been proposed that can be considered potential enablers for the implementation of the Model Web vision. The most cited are: Virtual (Research) Environments, (Jupyter) Notebooks, and data cubes. Some of them can be seen as complementary technologies and easily contribute to a digital ecosystem that makes use of the MaaR framework solution.

Virtual Environments: Over the past decades several information technology initiatives have started to support what is now the Open Science vision. They resulted in digital infrastructures variously termed as: Collaborative e-Research Communities, Collaborative Virtual Environments, Collaboratories, Science Gateways, Virtual Organisations, Virtual Research Communities, Cyberinfrastructures, Virtual Research Environments, Virtual Laboratories (Carusi and Reimer, 2010). Although they are not synonyms, they all share the idea of facilitating collaborative research –at least in some respects. As such, they are commonly focused on a research community (along with its narrower set of requirements, including usability and user-friendliness) that often drives the effort to the development of specialized and closed systems, based on Model-as-a-Tool approach.

Jupyter Notebooks: Project Jupyter (Jupyter Community) is a spin-off of the original IPython software, with the objective of extending its principle "across dozens of programming languages". Among the Project Jupyter products, the Jupyter Notebook is an interactive environment where users can write code, interactively run it, and visualize results. Jupyter Notebook uses include: "data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more". Jupyter Notebooks and similar solutions are now empowering cloud platforms and Virtual Research Environments like Google Colab (Google Colab). They are excellent tools for rapid development and (interactive) documentation, but as for Virtual Environments these developments are suited to the specific needs of a community. It would be interesting to integrate these notebook solutions into a MaaR framework. This would be relatively easy since they are designed as Web applications working in a Web environment. The integration could also leverage the on-going efforts on the use of Jupyter notebooks for environmental modelling (e.g., in the hydrology community (Choi et al., 2021)) and for interacting with data cubes (e.g., in the Earth Observations community (Gomes et al., 2020)).

Data Cubes: data cubes recently emerged as a promising solution to data and model integration and sharing. A data cube is a system allowing to ingest datasets in a multi-dimensional array of values and access and process them through a Web interface or API (Giuliani et al., 2019). Data cubes allow serving Analysis Ready Data (ARD), because (in the data cube) all dataset values are projected on the same coordinate reference system and uniformly pre-processed. Google Earth Engine, Rasdaman, and Open Data Cube are well-known examples of technologies implementing the data cube principle. With some specificities they all enable user to directly perform computations on data cubes and store processing procedures. Although data cubes considerably lower entry barriers for data users, they also have significant drawbacks and limitations (Nativi et al., 2017). First, ARD is a potentially misleading concept since readiness depends on the intended use of data. Therefore, it would be more correct to distinguish among different usages: ARD for monitoring and assessing landscape change, ARD for humanitarians, ARD for flash floods, etc. The different types of "readiness" may have a great impact –for example, for some user data are ready when cloud cover is removed, but, for others, clouds are the subject of their study and should not be removed. Even when pre-processing is limited to data projection, interoperability can be an issue since data cubes covering different areas may need adopting different coordinate reference systems –e.g., azimuthal projection for polar regions, and UTM for other regions. These issues clearly limit the usability of a data cube beyond its intended design. Therefore, for data sharing, data cubes can be considered an evolution of the traditional concept of data servers, providing advanced functionalities and increased performances through the ingestion and transformation process. However, they still have the overall interoperability issues of traditional data servers. Concerning scientific models, data cubes have excellent performances for specific use-cases (e.g., time series processing) due to the reorganization of data, but no specific improvement on other use-cases. They adopt a MaaS approach and moving a model from a data cube to another is not straightforward. Data cubes can be integrated as data servers in the service-oriented subsystem of a MaaR framework, but a full exploitation of their capabilities is possible only if its data and models are fully exposed as resources.

5. Discussion

To share and run computational models, the proposed MaaR framework presents several opportunities and some challenges, which can be summarized in a Strength-Weaknesses-Opportunities-Threats (SWOT) analysis.

5.1. Strengths

Clear separation-of-concerns: 'User-driven', 'user-centric', 'codesign', etc. are widespread terms commonly utilized to indicate the active involvement of different stakeholders in the design and implementation of a system. However, often, a clear identification and understanding of system users lack. Not all users are equal, end-users are not (usually) application providers who are themselves different from service providers. They have different expertise and needs, and some users may not be aware of the ultimate technological solutions. Therefore, to effectively engage users in a system design, it is useful to apply the software pattern called "separation-of-concern": each user can enter the system at her/his own level of expertise and abstraction. The proposed multi-style architecture allows data and model providers to easily focus on publishing resources on the Web, while service providers can only deal with building complex systems for orchestrating access to resources; moreover, application developers are able to create Web applications using APIs, and, finally, end users must only interact with the system through the well-known and user-friendly navigation paradigm.

Low entry barrier: In keeping with the separation-of-concern pattern, the proposed MaaR framework has low entry barriers for the different user categories. Data and model providers can easily publish their resources by using a widespread platform –such as an existing data server or source code repository, or a simple Web server. Service providers can build their services that orchestrate the access to the published resources and host them on a commercial or public platform. Software developers can build Web applications by using existing services, often accessible through dedicated APIs. Finally, end users need only a browser to run complex simulations which are hidden behind a Web interface.

Extensibility: Due to its open architecture, which does not impose major constraints for Web protocols compliance, the proposed MaaR framework is greatly extensible. In principle, everyone may enrich the resulting Model Web, by adding resources, annotating those already existing, providing services to create added value, and building applications for specific users. This extensibility makes the MaaR framework an ideal candidate for building a digital ecosystem –around the concept of Model Web. As in any successful ecosystem, the different categories of users can pursue their own advantage by sometimes adopting a collaborative approach and other times a competitive one, but, in any case, enriching the digital environment and increasing the overall service of the digital ecosystem – which is the generation of knowledge for informed decision-making.

Viability: Due to its own disperse nature, the MaaR framework is subject to changes, over time, for both internal reasons (i.e. changes of the constituent digital infrastructures and software systems) and external reasons (i.e. changes of the societal and policy needs, and the many technological revolutions). Without any control, those changes may result disruptive, making impossible for the framework to pursue its intended objectives. In digital ecosystems, the essential role of the invariants consists in providing the capability to detect changes and respond to those threatening the system in a manner consistent with the objectives. Clearly identifying a major invariant as its multi-style architecture, the MaaR framework adopts such approach, making it viable – i.e., a system that can sustain itself over time.

5.2. Weaknesses

Multi-style architectures are fragile: The proposed MaaR framework is based on a multi-style architecture. It assures the benefits of resource-oriented (e.g., user-friendliness, openness) and service-oriented architectures (e.g., capabilities), introducing an architectural constraint which imposes to maintain an interface for separating the two subsystems. This required invariant needs some governance mechanism to intervene against violations.

Initial complexity of the digital environment: To result user-friendly and have low entry barriers, the proposed framework must already rely on a digital environment of online services, which complement and give value to its functionalities. For instance, a model provider can easily share a model as a source code because a system administrator can annotate the model with instructions and build a Docker container; moreover, a service provider can offer an orchestration service for building the Docker container and deploy it on a cloud platform, etc. Then, to make everything work, an initial level of complexity of the digital environment must be already present. This means that some services should exist to make it possible to attract a critical mass of users of different categories and trigger the coevolution process. Fig. 6 shows what could be the minimal set of components to be available in the earliest stage of a digital ecosystem for a Model Web – i.e., data, model, and cloud brokers would allow to expose an open virtual platform for data and model sharing.

Data and models availability: Lack of online data and models is a general drawback impacting any potential sharing system. However, this possible barrier is more important for a MaaR framework as it requires (or at least encourage) the sharing of as much information as possible on data and models – to understand whether they are exploitable resources for various use-cases implementation. To be effective, the MaaR framework assumes that different kinds of data and models are available and accessible on the Web. The MaaR framework tries lowering interoperability barriers, but there are still many other barriers to data and model sharing: attitudinal barriers (lack of awareness about the importance of sharing for a more effective science); financial barriers (lack of time and effort/funds to share resources); legal barriers (lack of clear rules and policies on digital resource sharing); and technical barriers (lack of portability of very complex digital resources –e.g., scientific models).

Semantic interoperability: There is a weakness dealing with the sound orchestration of the many different digital resources (available on the Web) to obtain a useable piece of information or knowledge. Semantic interoperability would significantly help addressing this issue, but the semantic description of data and models is not yet mature enough. Therefore, the control of scientific soundness of resources orchestration must be done, by expert, in an offline way – e.g., by defining a-priori workflow to be implemented through the MaaR framework or by assessing the orchestration chain and result a-posteriori. In summary, the MaaR is an instrument to be supervised.

5.3. Opportunities

Beside the system strengths and weaknesses which are related to the community to be built around it, the proposed MaaR framework can leverage relevant opportunities in general:

Multi-disciplinarity: the global (societal, economic, and environmental) changes that we are experimenting require to face great challenges (disaster resilience, climate change adaption, sustainable development, etc.) which are inherently multidisciplinary. This requires integrating data and models to implement sound processes for knowledge generation. We can expect an extremely heterogeneous set of resources –data encoded in different formats, models built in old programming languages, resources described and annotated according to heterogeneous ontologies and vocabularies. It is very unlikely that a single solution (i.e., a tool or a standard) could be adopted to link them all. The best approach is to accept their heterogeneity, enrich their description (as much as possible), and adopt a mediation approach. The proposed MaaR framework aims to provide an architecture that makes possible to share and use heterogeneous resources in a common environment.

Digital transformation: the technologies that have supported the digital transformation of society provide the foundation for building a MaaR framework – e.g., cloud platforms offer an affordable and ubiquitous access to storage and computing services, and containerization technologies provide the possibility to harmonize software environment for model running. More specifically, the tools developed for the geospatial domain (i.e., data and models brokers, datacubes, etc.) potentially provide the core components for the digital environment in which a geospatial digital ecosystem can evolve.

5.4. Threats

Threats for the proposed MaaR framework are mostly related to governance aspects. The MaaR framework can be the basis for a viable ecosystem, but as for any ecosystem, there is the need to set up an initial digital environment, reach a critical mass of users and maintain that. The setup of the digital environment is mostly a matter of time, effort, and money – the utilization of mature technologies and widespread solutions could minimize them. Instead, reaching and maintaining a critical mass of users requires to face several specific challenges including a) to make clear the overall value of the digital ecosystem; b) to make clear the value of the digital ecosystem for each user category; c) to overcome potential competition with other values – e.g., what if some stakeholders see model sharing, with closed tools, as a potential market/funding opportunity? The last point is important because the MaaR framework tries to replicate the Web approach moving from static to dynamic resources. Anyway, when the Web was born there was no other option, and no stakeholder had any idea or opportunity to create something alternative. Today, the Web is in place as a resource sharing platform, and models sharing can follow different directions, depending on conflicting interests.

6. Conclusions and future directions

This document provides a high-level description of a flexible framework for sharing and running computational models in a distributed system. Unlike many existing examples of Web-based sharing systems, it adopts a top-down approach highlighting the importance of the choice of a clear architectural style expressed through well-defined constraints that assure a coherent evolution preserving the relevant system characteristics. As such, it aims at assuring the viability of Web-based systems leveraging innovative technologies (e.g., incorporating them through services and resource gateways as in Fig. 6), being open to other communities (e.g., allowing to introduce third-party brokers), and fully supporting Open Knowledge demands (e.g., with components dedicated to semantics for replicability and reusability). This is essential to support the creation of a Model Web from perspective of a digital ecosystem where requirements, actors, and even objectives can change over time.

Although the proposed framework is mainly aimed at models relevant to environmental applications, there are no specific constraints preventing its adoption with other types of models or for other application domains, including socio-economic models or algorithms for generating composite indices.

The proposed framework is based on the Model-as-a-Resource (MaaR) approach, where a model is considered an information resource, which is an element of interest in the information space. This means that a model is potentially something valuable in many possible different scenarios. This contrasts with the Model-as-a-Service (MaaS) approach, where a model is seen as an executable resource that plays a valuable role only in a predefined scenario. The concept of the model as a resource fits better into the Open Science view. It also responds more to the needs of a transparent decision-making process, in which end users need not only the result of model execution but also the information that characterizes the model itself and the context in which it can be used. A MaaR can be considered a self-documenting resource.

The proposed MaaR framework is based on the REST architectural style, which is the original architectural style of the Web. The choice is based on the proven capability of such a style to support scalability and evolvability. However, we recognized that running a model has high-level requirements and that non-expert users need assistance. Therefore, the proposed MaaR framework adopts a multi-style architecture, integrating a RESTful subsystem and a service-oriented one. Dedicated components (gateways) help mapping service interactions with RESTful applications. It is essential to clearly define the boundaries between the service-oriented and the resource-oriented sub-systems.

We argued that the multi-style architecture of the MaaR framework fits perfectly to realize the scenario of ever-changing digital ecosystems. For this reason, the framework is an important tool to realize the original vision of a Model Web of representational models driven by theory and data. There are still some open challenges, however these are not mainly of a technical nature, but related to the governance aspects of how to build an initial digital environment and how to reach a critical mass of participating users.

For application developers, the multi-style architectural constraint means: "build your applications as lightweight Web applications". As a result, this suggested designing proper interfaces to service providers for supporting developers –and facilitate mashups. Moreover, these recommendations are perfectly plausible for their targets for reasons beyond their architectural nature. Application developers may follow the recommendation to "build your applications as lightweight Web applications" not because it is an architectural constraint, but because it is a way to build a user-friendly application and so make it a success. Service providers may want to provide APIs just because they could make their product more appealing for developers.

Once the digital ecosystem is established and active, those who violate the constraint (e.g.; by publishing resources with closed tools, offering services with an integrated client, or building applications with non-Web clients) would put themselves out of the ecosystem; their resources, services, or applications could not be integrated with the others in the ecosystem – likely, they would give small value to providers and users.

Future works include the study of the application of the MaaR framework to the specificities of: a) data-driven models based on Machine Learning to support their full lifecycle (from the training phase to the deployment) and b) digital twins to support their connectivity with input and output data streams.

CRediT authorship contribution statement

Paolo Mazzetti: Writing – review & editing, Writing – original draft, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization. **Stefano Nativi:** Writing – review & editing, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

The research leading to these results has received funding under a research contract from the DG Joint Research Centre of the European Commission (in the framework of the "Study on Assessing & Proposing an Implementation Architecture" of the Destination Earth initiative: an AA between JRC and DG CNECT), from ESA under contract n. 4000138128/22/I/AG (DAB4GPP), from the European Union's Digital Europe Programme under Grant Agreement n. 101083927 (GREAT).

The authors would like to thank Michael Lutz and Max Craglia (from DG JRC) for the continued discussion we have had on these issues and Nicholas Spadaro (from DG JRC), Mattia Santoro, and Massimiliano Olivieri (from CNR-IIA) for the help in implementing proof-of-concepts.

References

Annoni, A., et al., 2023. Digital earth: yesterday, today, and tomorrow. Int. J. Digital Earth 16 (1), 1022–1072. https://doi.org/10.1080/17538947.2023.2187467.
Barth, A., 2011. HTTP State Management Mechanism," Internet Engineering Task Force, Request for Comments RFC 6265. https://doi.org/10.17487/RFC6265.
Barton, C.M., et al., 2020. Call for transparency of COVID-19 models. Science 368 (6490), 482–483. https://doi.org/10.1126/science.abb8637.
Bastin, L., et al., 2013. Managing Uncertainty in Integrated Environmental Modelling: the UncertWeb Framework, vol. 39. Environmental Modelling & Software, pp. 116–134. https://doi.org/10.1016/j.envsoft.2012.02.008.
Baumann, P., 2010. The OGC web coverage processing service (WCPS) standard. GeoInformatica 14 (4), 447–479. https://doi.org/10.1007/s10707-009-0087-2.
Berners-Lee, T., Hendler, J., Lassila, O., 2001. The Semantic Web. Scientific American.
Berners-Lee, T., Fielding, R.T., Masinter, L.M., 2005. "Uniform Resource Identifier (URI): Generic Syntax," Internet Engineering Task Force. Request for Comments RFC 3986. https://doi.org/10.17487/RFC3986.
Boldrini, E., Nativi, S., Pecora, S., Chernov, I., Mazzetti, P., 2022. Multi-scale hydrological system-of-systems realized through WHOS: the brokering framework. Int. J. Digital Earth 15 (1), 1259–1289. https://doi.org/10.1080/17538947.2022.2099591.
Braun, M.L., Ong, C.S., 2018. Open science in machine learning. In: Stodden, V., Leisch, F., Peng, R.D. (Eds.), Implementing Reproducible Research, first ed. Chapman and Hall/CRC, Boca Raton, FL.
Car, N., Cox, S., Fitch, P., 2015. Associating Uncertainty with Datasets Using Linked Data and Allowing Propagation via Provenance Chains. Apr, p. 4392.
Carlos, F., De Salvo, P., Queiroz, G.R., Franziskakis, F., Glaves, H., 2022. Sharing and Preserv. GEO Commun. Appl. Through the GEO Knowledge Hub 2022. IN41A-02.
Carusi, A., Reimer, T., 2010. Virtual Research Environment Collaborative Landscape Study [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.404.6517&rep=rep1&type=pdf.
Chen, M., et al., 2020. Position paper: open web-distributed integrated geographic modelling and simulation to enable broader participation and applications. Earth Sci. Rev. 207, 103223 https://doi.org/10.1016/j.earscirev.2020.103223.
Choi, Y.-D., et al., 2021. Toward Open and Reproducible Environmental Modeling by Integrating Online Data Repositories, Computational Environments, and Model Application Programming Interfaces, vol. 135. Environmental Modelling & Software, 104888. https://doi.org/10.1016/j.envsoft.2020.104888.
CNR, "GEODAB - GEO Discovery and Access Broker," GEODAB. [Online]. Available: https://www.geodab.net/.
Erl, T., 2005. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, NJ.
Essawy, B.T., et al., 2018. Integrating Scientific Cyberinfrastructures to Improve Reproducibility in Computational Hydrology: Example for HydroShare and GeoTrust, vol. 105. Environmental Modelling & Software, pp. 217–229. https://doi.org/10.1016/j.envsoft.2018.03.025.
European Commission, 2018. Directorate General for Research and Innovation., *Turning FAIR into Reality: Final Report and Action Plan from the European Commission Expert Group on FAIR Data*. LU: Publications Office. Feb. 03, 2021. [Online]. Available: https://data.europa.eu/doi/10.2777/1524.
European Commission Directorate-General for Research & Innovation, "Guidelines on FAIR Data Management in Horizon 2020." Jul. 2016. [Online]. Available: http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf..
Fecher, B., Friesike, S., 2014. Open science: one term, five schools of thought. In: Bartling, S., Friesike, S. (Eds.), Opening Science. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-00026-8.
R. T. Fielding, "REST APIs must be hypertext-driven," Untangled. [Online]. Available: https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven..
Fielding, R.T., 2000. Architectural styles and the design of network-based software architectures. Nov. 21, 2019. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
Fielding, R.T., Taylor, R.N., 2002. Principled design of the modern Web architecture. ACM Trans. Internet Technol. 2 (2), 115–150. https://doi.org/10.1145/514183.514185.
Fielding, R.T., et al., 2017. Reflections on the REST architectural style and 'principled design of the modern web architecture' (impact paper award). In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, in ESEC/FSE 2017. Association for Computing Machinery, New York, NY, USA, pp. 4–14. https://doi.org/10.1145/3106237.3121282.
Flaishans, J., et al., 2016. Environmental Models as a Service: Enabling Interoperability through RESTful Endpoints and API Documentation. International Congress on Environmental Modelling and Software [Online]. Available: https://scholarsarchive.byu.edu/iemssconference/2016/Stream-A/18.
Foerster, T., Brühl, A., Schäffer, B., 2011. RESTful web processing service. In: Proceedings of the 14th AGILE International Conference on Geographic Information Science [Online]. Available: https://agile-online.org/conference_paper/cds/agile_2011/contents/pdf/shortpapers/sp_137.pdf.
Frigg, R., Hartmann, S., 2020. Models in science. In: Zalta, E.N. (Ed.), *The Stanford Encyclopedia Of Philosophy*, Spring 2020, Metaphysics Research Lab. Stanford University. Jul. 06, 2021. [Online]. Available: https://plato.stanford.edu/archives/spr2020/entries/models-science/.
Geller, G.N., Melton, F., 2008. Looking forward: applying an ecological model web to assess impacts of climate change. Biodiversity 9 (3–4), 79–83. https://doi.org/10.1080/14888386.2008.9712910.
Geller, G.N., Turner, W., 2007. The model web: a concept for ecological forecasting. In: 2007 IEEE International Geoscience and Remote Sensing Symposium. IEEE, Barcelona, Spain, pp. 2469–2472. https://doi.org/10.1109/IGARSS.2007.4423343.
GEO, 2021. GEO statement on open knowledge [Online]. Available: https://www.earthobservations.org/documents/geoweek2021/GEO-17-4.1_GEO%20Statement%20on%20Open%20Knowledge.pdf.
Giuliani, G., Masó, J., Mazzetti, P., Nativi, S., Zabala, A., 2019. Paving the way to increased interoperability of earth observations data cubes. Data 4 (3), 113. https://doi.org/10.3390/data4030113.
Gomes, V., Queiroz, G., Ferreira, K., 2020. An overview of platforms for big earth observation data management and analysis. Rem. Sens. 12 (8), 1253. https://doi.org/10.3390/rs12081253.
Google Colab." Accessed: Mar. 30, 2023. [Online]. Available: https://research.google.com/colaboratory/faq.html..
Granell, C., Díaz, L., Schade, S., Ostländer, N., Huerta, J., 2013. Enhancing Integrated Environmental Modelling by Designing Resource-Oriented Interfaces, vol. 39. Environmental Modelling & Software, pp. 229–246. https://doi.org/10.1016/j.envsoft.2012.04.013.
Group on Earth Observations, "GEO Knowledge Hub," GEO. [Online]. Available: https://gkhub.earthobservations.org/.

Imbert, C., 2017. Computer simulations and computational models in science. In: Bertolotti, T., Magnani, L. (Eds.), Springer Handbook of Model-Based Science, first ed. Springer Berlin Heidelberg, New York, NY.

ISO, "ISO/IEC 10746-1:1998(en), Information technology — Open Distributed Processing — Reference model: Overview — Part 1." Accessed: Jul. 20, 2021. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:10746:-1:ed-1:v1:en..

Jupyter Community, "Project Jupyter." Accessed: Aug. 04, 2021. [Online]. Available: https://www.jupyter.org.

Laniak, G.F., et al., 2013. Integrated Environmental Modeling: A Vision and Roadmap for the Future, vol. 39. Environmental Modelling & Software, pp. 3–23. https://doi.org/10.1016/j.envsoft.2012.09.006.

Lastovetsky, A.L., Gaissaryan, S.S., 1994. An algebraic approach to semantics of programming languages. Theor. Comput. Sci. 135 (2), 267–288. https://doi.org/10.1016/0304-3975(94)00022-0.

Lloyd, E.A., Winsberg, E. (Eds.), 2018. Climate Modelling: Philosophical and Conceptual Issues, first ed. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-65058-6. Imprint: Palgrave Macmillan, 2018.

Mazzetti, P., Nativi, S., Caron, J., 2009. RESTful implementation of geospatial services for Earth and Space Science applications. Int. J. Digital Earth 2 (S1), 40–61. https://doi.org/10.1080/17538940902866153.

Mazzetti, P., et al., 2022. Knowledge formalization for earth science informed decision-making: the GEOEssential knowledge base. Environ. Sci. Pol. 131, 93–104. https://doi.org/10.1016/j.envsci.2021.12.023.

Melnikov, A., Fette, I., 2011. "The WebSocket Protocol," Internet Engineering Task Force, vol. 6455. Request for Comments RFC. https://doi.org/10.17487/RFC6455.

Moschovakis, Y.N., 1993. Sense and Denotation as Algorithm and Value," *Logic Colloquium '90*. ASL Summer Meeting, Helsinki, pp. 210–249.

Nativi, S., Craglia, M., 2021. Destination Earth: Ecosystem Architecture Description. LU: Publications Office of the European Union. Aug. 06, 2021. [Online]. Available: https://data.europa.eu/doi/10.2760/08093.

Nativi, S., et al., 2004. Differences among the data models used by the geographic information systems and atmospheric science communities. In: Proceedings American Meteorological Society - 20th Interactive Image Processing Systems Conference [Online]. Available: https://ams.confex.com/ams/84Annual/techprogram/paper_73229.htm.

Nativi, S., et al., 2007. Predicting the impact of climate change on biodiversity: a GEOSS scenario. In: The Full Picture. Group on Earth Observations, pp. 262–278.

Nativi, S., et al., 2009a. A technology framework to analyse the Climate Change impact on biodiversity species distribution. In: EGU General Assembly Conference Abstracts, p. 3436.

Nativi, S., Mazzetti, P., Saarenmaa, H., Kerr, J., Tuama, É.Ó., 2009b. Biodiversity and climate change use scenarios framework for the GEOSS interoperability pilot process. Ecol. Inf. 4 (1), 23–33. https://doi.org/10.1016/j.ecoinf.2008.11.002.

Nativi, S., Mazzetti, P., Geller, G.N., 2012. Environmental model access and interoperability: the GEO Model Web initiative. Environ. Model. Software. https://doi.org/10.1016/j.envsoft.2012.03.007.

Nativi, S., Craglia, M., Pearlman, J., 2013. Earth science infrastructures interoperability: the brokering approach. IEEE J. Sel. Top. Appl. Earth Obs. Rem. Sens. 6 (3), 1118–1129. https://doi.org/10.1109/JSTARS.2013.2243113.

Nativi, S., Mazzetti, P., Craglia, M., 2017. A view-based model of data-cube to support big earth data systems interoperability. Big Earth Data 1 (1–2), 75–99. https://doi.org/10.1080/20964471.2017.1404232.

Nativi, S., Mazzetti, P., Craglia, M., 2021. Digital ecosystems for developing digital twins of the earth: the destination earth case. Rem. Sens. 13 (11), 2119. https://doi.org/10.3390/rs13112119.

Nielsen, H., Fielding, R.T., Berners-Lee, T., 1996. Hypertext Transfer Protocol – HTTP/1.0," Internet Engineering Task Force, Request for Comments RFC 1945. https://doi.org/10.17487/RFC1945.

Ninyerola, M., et al., 2014. QualityML: a Dictionary for Quality Metadata Encoding, 10452.

NOAA, "ERDDAP," ERDDAP Web Site. Accessed: Aug. 04, 2021. [Online]. Available: https://www.ncei.noaa.gov/erddap/index.html..

OASIS Open, 2006. Reference model for service oriented architecture 1.0 [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

OASIS Open, "Standards," OASIS Open. [Online]. Available: https://www.oasis-open.org/standards/..

Odifreddi, P., 1989. *Classical recursion theory: the theory of functions and sets of natural numbers*, 2 vols. In: Studies in Logic and the Foundations of Mathematics, vol. 125. North-Holland, Amsterdam ; New York : New York, N.Y., USA, p. 143.

OGC, "OGC WPS 2.0.2 Interface Standard Corrigendum 2." Mar. 05, 2015. [Online]. Available: http://docs.opengeospatial.org/is/14-065/14-065.html..

OGC, 2009. "Uncertainty Markup Language (UnCertML)." Apr. 08 [Online]. Available: https://portal.ogc.org/files/?artifact_id=33234.

OGC, OpenGIS Web Processing Service. Jun. 08, 2007. [Online]. Available: https://portal.ogc.org/files/?artifact_id=24151..

Roy Fielding on Versioning, Hypermedia, and REST, InfoQ. Accessed: Jun. 25, 2021. [Online]. Available: https://www.infoq.com/articles/roy-fielding-on-versioning/..

Salvendy, G., Karwowski, W. (Eds.), 2010. Introduction to Service Engineering. John Wiley & Sons, Hoboken, N.J.

Sansone, S.-A., et al., 2019. FAIRsharing as a community approach to standards, repositories and policies. Nat. Biotechnol. 37 (4) https://doi.org/10.1038/s41587-019-0080-8. Art. no. 4.

Santoro, M., Nativi, S., Mazzetti, P., 2016. Contributing to the GEO Model Web Implementation: A Brokering Service for Business Processes, vol. 84. Environmental Modelling & Software, pp. 18–34. https://doi.org/10.1016/j.envsoft.2016.06.010.

Santoro, M., Mazzetti, P., Nativi, S., 2020. The VLab framework: an orchestrator component to support data to knowledge transition. Rem. Sens. 12 (11), 1795. https://doi.org/10.3390/rs12111795.

Stasch, C., Jones, R., Cornford, D., Kiesow, M., Williams, M., Pebesma, E., 2012. "Representing Uncertainties in the Sensor Web," Presented at the Sensing a Changing World 2012. Wageningen, The Netherlands [Online]. Available: https://www.wur.nl/upload_mm/e/4/9/5d9cd704-8472-421e-acb7-ab9723bfc5e7_Stasch_etal.pdf.

Tarboton, D.G., et al., 2024. HydroShare Retrospective: Science and Technology Advances of a Comprehensive Data and Model Publication Environment for the Water Science Domain, vol. 172. Environmental Modelling & Software, 105902. https://doi.org/10.1016/j.envsoft.2023.105902.

Tolk, A., Diallo, S., Turnitsa, C., 2007. Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systems engineering. J. Syst., Cybern. Informatics 5 (5), 65–74.

Villa, F., Balbi, S., Athanasiadis, I.N., Caracciolo, C., 2017. Semantics for interoperability of distributed data and models: foundations for better-connected information. F1000Res 6, 686. https://doi.org/10.12688/f1000research.11638.1.

W3C, 2004. Architecture of the world wide web, volume one. Nov. 21, 2019. [Online]. Available: https://www.w3.org/TR/webarch/.

W3C, "RDFa Core 1.1 - Third Edition." Accessed: Jun. 25, 2021. [Online]. Available: https://www.w3.org/TR/rdfa-core/.

Watada, J., Roy, A., Kadikar, R., Pham, H., Xu, B., 2019. Emerging trends, techniques and open issues of containerization: a review. IEEE Access 7, 152443–152472. https://doi.org/10.1109/ACCESS.2019.2945930.

Web Hypertext Application Technology Working Group (WHATWG), "HTML Standard." Accessed: Aug. 08, 2022. [Online]. Available: https://html.spec.whatwg.org/multipage/.

Wilkinson, M.D., et al., 2016. The FAIR Guiding Principles for scientific data management and stewardship. Sci. Data 3 (1), 160018. https://doi.org/10.1038/sdata.2016.18.

Williams, M., Cornford, D., Bastin, L., 2008. "Describing and Communicating Uncertainty within the Semantic Web," Presented at the 7th International Semantic Web Conference. Karlsruhe, Germany [Online]. Available: http://eprints.aston.ac.uk/10038/.

52North, 52North 52° North, "WPS 2.0 REST API TAMIS," 52north website. [Online]. Available: https://52north.github.io/tamis-rest-api/..

ISO/TC 211 - Geographic information/Geomatics, ISO. Accessed: Dec. 11, 2023. [Online]. Available: https://www.iso.org/committee/54904.html.

OGC Standards, OGC. Accessed: Dec. 11, 2023. [Online]. Available: https://www.ogc.org/standards/..

QualityML. Accessed: Aug. 03, 2021. [Online]. Available: http://www.qualityml.org/..