OKG-Soft: An Open Knowledge Graph with Machine Readable Scientific Software Metadata

Daniel Garijo
Information Sciences Institute,
University of Southern California
Marina del Rey, CA, USA
dgarijo@isi.edu

Maximiliano Osorio
Information Sciences Institute,
University of Southern California
Marina del Rey, CA, USA
mosorio@isi.edu

Deborah Khider
Information Sciences Institute,
University of Southern California
Marina del Rey, CA, USA
khider@isi.edu

Varun Ratnakar
Information Sciences Institute,
University of Southern California
Marina del Rey, CA, USA
varunr@isi.edu

Yolanda Gil
Information Sciences Institute,
University of Southern California
Marina del Rey, CA, USA
gil@isi.edu

Abstract—**Scientific software is crucial for understanding, reusing and reproducing results in computational sciences. Software is often stored in code repositories, which may contain human readable instructions necessary to use it and set it up. However, a significant amount of time is usually required to understand how to invoke a software component, prepare data in the format it requires, and use it in combination with other software. In this paper we introduce OKG-Soft, an open knowledge graph that describes scientific software in a machine readable manner. OKG-Soft includes: 1) an ontology designed to describe software and the specific data formats it uses; 2) an approach to publish software metadata as an open knowledge graph, linked to other Web of Data objects; and 3) a framework to annotate, query, explore and curate scientific software metadata. OKG-Soft supports the FAIR principles of findability, accessibility, interoperability, and reuse for software. We demonstrate the benefits of OKG-Soft with two applications: a browser for understanding scientific models in the environmental and social sciences, and a portal to combine climate, hydrology, agriculture, and economic software models.**

Keywords—software metadata, software registries, FAIR, knowledge graphs, software composition, software interoperability

I. Introduction

Software is a key product of scientific research, as it can be used to understand and reproduce the findings reported in a publication (e.g., by rerunning a hydrology model, a genome sequence analysis or testing a trained machine learning model). The importance of software is increasingly recognized [1], with publishers and community initiatives encouraging researchers to make their software openly available to others.[1]

Scientific software created by scientists should be appropriately documented and curated to facilitate reuse by other researchers. Code repositories such as GitHub [2] or BitBucket[3] provide the means to store and version code, while software container repositories such as DockerHub[4] capture the execution environment required to run software. However, there is usually a lack of important information that makes software difficult to discover and reuse, such as descriptions of the main features of the software, unambiguous usage instructions, incomplete sample data, etc. Moreover, when this kind of information is present, it is not machine readable, so it is hard to develop tools to facilitate those tasks for users.

A major barrier to reuse is the time and effort required to understand how to run scientific software. Researchers need to understand how to prepare data for software, how to invoke it, and how to interpret the results produced after its execution. Despite the desire to use standards, different software codes operate with heterogeneous data formats. Studies have shown that scientists spend between 60% and 80% performing data preparation steps when composing software together in scientific workflows [2]. This problem is compounded in numerous applications that require combining software that has been developed by independent third parties. For example, combining software for population genomics with another for genomic network analysis, or combining a hydrology model software with an agriculture model.

While many common formats and standards have been proposed, [5] this alone will not solve the interoperability problem. First, there are still quite diverse standards for the same kind of data, and a given software package usually adopts only one. A researcher that wants to use the software with data from more than one source often needs to understand that particular format adopted by the software, then write code to do the necessary transformations. Second, when composing different software, it is often the case that the data formats are different. In some cases, the software and standards may have been developed by different communities (e.g., hydrology and agriculture). Addressing these challenges requires that scientific software is described with sufficient details about the data used, in terms of both format and content. And if these representations are machine readable then it would be possible to develop tools to do data transformations automatically.

In this paper we present OKG-Soft, a framework to capture and publish machine-readable software metadata. OKG-Soft builds on OntoSoft [3], [4], our previous work to capture scientific software metadata, and expands it with machine readable descriptions of the expected contents of inputs and outputs of software. OKG-Soft has three main novel contributions:

1- *A modular ontology to describe software and its associated input and output metadata.*

[1] https://paperswithcode.com
[2] github.com/
[3] https://bitbucket.org/
[4] hub.docker.com/
[5] https://frictionlessdata.io/specs/

This work was funded by the Defense Advanced Research Projects Agency with award W911NF-18-1-0027, and the National Science Foundation with award ICER-1440323

2- *An approach to publish software metadata within an open knowledge graph, and linking it to the Web of Data following Linked Data principles* [5], [6].

3- *A framework designed to populate, query, explore, and curate software metadata.*

We demonstrate the benefits of our approach by capturing metadata for software from environmental and social sciences, including software models from climate, hydrology, economy and agriculture, and by showing how this metadata can be used to explore, understand, and compose diverse software.

Our approach with OKG-Soft captures and publishes machine-readable metadata in support of the FAIR principles of findability, accessibility, interoperability, and reuse for software [7].

The rest of the paper is structured as follows. Section 2 describes related work for capturing, storing and accessing software metadata. Section 3 explains the insights of OKG-Soft, i.e., the rationale behind the ontologies we propose and reuse; how the knowledge graph is populated, linked and enriched with existing knowledge bases; and how different types of developers can access the content of OKG-Soft through our proposed APIs. Section 4 presents how we have validated our approach with a series of queries to gather insight about the software entries in the knowledge graph, together with two showcase applications, one for exploring software model metadata and another one for composing software. Finally, Section 5 concludes the paper discussing our current efforts and future work.

II. Related Work

In this section we discuss existing ontologies for capturing different aspects of software metadata, along with systems that facilitate describing, capturing and sharing software.

A. Ontologies for Capturing Software Metadata

A number of ontologies have been proposed to describe software at different levels of granularity. In our previous work we presented OntoSoft [8] and OntoSoft-VFF [4], which capture scientific software metadata from a scientist's perspective through a series of questions they are familiar with. In this work we extended OntoSoft to expose additional metadata, context and semantic relationships between entities associated with software. This includes the expected contents of software input and output files, which help determining compatible software components.

The CodeMeta project[6] [9] is a community driven effort that presents a generic crosswalk from common terms used by code repositories to describe software (e.g., pointing to the code repository, readme file instructions, license, metadata, etc.). CodeMeta includes a mapping to OntoSoft and extends the Schema.org vocabulary, [7] which has been widely adopted by the community. This vocabulary does not semantically structure the contents of software (i.e., inputs, outputs and executable information), but we have reused in our work to incorporate generic metadata about software.

The Description of a Project Ontology (DOAP) [8] is designed to describe software projects, emphasizing the description of the organization of software (issues, bug tracking, wiki discussions, etc.). While this effort is related to software, it goes beyond the scope of this paper.

The Core Software Ontology (CSO)[9] and Core Ontology of Software Components (COSC) [10] extend the DOLCE [11] upper ontology to describe software libraries and web services in detail. CSO specifies concepts related to software and data, and includes both software components and services. COSC extends CSO to define software components further. An interesting aspect of these ontologies is that inputs and outputs of software are defined as roles, played by different types of data. Roles are adopted for addressing policies, which have to be specified by users. Such formalizations require users to understand logic inference in ways that makes the vocabularies difficult to reuse. The link between software inputs and their expected contents is not modeled in CSO.

In the bioinformatics domain, the Software Ontology (SWO)[10] extends some of the Open Biomedical Ontologies, such as the Basic Formal Ontology [12] and the Relations Ontology [11] to describe software relationships. SWO also extends the EDAM Ontology [13], which describes common data types and formats used in bioinformatics, linking them to a taxonomy of software. In addition, SWO describes a thorough taxonomy of programming languages, but defines them as classes instead of instances. SWO does not describe the expected contents of inputs, outputs or formats.

Finally, other ontologies have been designed to address specific aspects of software. For example, the DockerPedia ontology [14] captures executable information of software containers, such as their installed packages and potential vulnerabilities. DockerPedia builds on WICUS [15], an ontology to describe computational infrastructure for scientific experiments. We use these ontologies to inform our work in OKG-Soft.

B. Software Repositories and Software Metadata Registries

Scientists are increasingly using software repositories to store versions, test, integrate and disseminate their code. Repositories such as GitHub, GitLab [12] and BitBucket are perhaps the most widely used by the community, but don't usually hold much metadata besides license, creator and installation instructions. When releasing code, scientists may refer to other more specific platforms, such as Figshare [13] or Zenodo,[14] as they provide DOIs stating how to cite a particular code. Package repositories such as CRAN, [15] Pypi, [16] or Maven Central[17] focus on the ability to execute and import code, but not necessarily on its usability. Software container repositories such as DockerHub address the execution of software with complex dependencies, but usually lack metadata necessary for effectively understand software.

Software metadata registries focus on metadata descriptions of software, complementing code repositories which focus on the code. Software metadata registries may not store the code itself, but will likely have a pointer to a code repository to find it. Examples of software metadata registries

[6] https://github.com/codemeta
[7] http://schema.org/
[8] http://usefulinc.com/ns/doap
[9] http://km.aifb.kit.edu/sites/cos/
[10] http://purl.obolibrary.org/obo/swo.owl
[11] http://purl.obolibrary.org/obo/ro.owl
[12] https://gitlab.com/
[13] https://figshare.com/
[14] http://zenodo.org/
[15] https://cran.r-project.org/
[16] https://pypi.org/
[17] https://search.maven.org/

are the Community Surface Dynamics Modeling System (CSDMS), which contains hundreds of codes for models for Earth surface processes [16], the Computational Infrastructure for Geodynamics, [18] which emphasizes metadata on how to perform model specific operations such as coupling and regridding; the Astrophysics Source Code Library (ASCL), which contains unambiguous code descriptions in astrophysics [17]; and OntoSoft [18], which describes software for geosciences. These software registries contain instructions on how to run software, but do not usually represent this information in a machine-readable manner to facilitate software reuse and composition.

Scientific gateways such as NanoHub [19] allow describing and finding software, even executing tools individually. Similarly, scientific workflow systems combine software together to represent larger analyses. Scientific workflows usually include many software codes from a particular domain, e.g., LONI Pipeline for neuroimaging genomics[20], GenePattern and Galaxy for genomics [21], and Taverna for bioinformatics services [22]. However, while automated workflow composition based on inputs and outputs of software has been researched (e.g., [23], [24]) there is not much work in describing how the contents of inputs and outputs may be related to each other.

III. OKG-Soft: A knowledge graph for scientific software

OKG-Soft is an open knowledge graph designed to represent software metadata in a machine-readable manner. OKG-Soft pays special attention to the description of inputs and outputs of software, in order to represent their expected formats and contents that can support automated data preparation and software composition. We organize OKG-Soft in three main components: 1) an ontology designed to describe software and capture machine-readable metadata; 2) an open knowledge graph of software descriptions that publishes this metadata and 3) a curation and exploitation framework to facilitate developers and domain scientists exploring and reusing the contents of OKG-Soft. We further describe each of these components below.

A. *Software Description Ontology*

We have developed the Software Description Ontology (with prefix sd), the core ontology we propose for representing entities in OKG-Soft. The latest version of the ontology is available and documented at https://w3id.org/okn/o/sd.

1) Development Methodology

The Software Description Ontology relies on our previous work in OntoSoft [8] and OntoSoft-VFF [4]. The main differences with previous work are highlighted in Figure 1, and consist of a simplification of the core model, an extension of its semantics to relate concepts instead of textual descriptions and the addition of variables and metadata to capture how software may be used in composition with other software. The ontology was developed in an iterative manner, expanding the requirements from [18]. A list of our complete requirements may be found in [25] .

We have favored the reuse of existing vocabularies and standards in our ontology development. We adopt Codemeta [9] to describe all basic software attribution terms, such as author, maintainer, funding, license, associated publications, etc. Codemeta is a community-driven initiative that uses Schema.org as core vocabulary to represent software (*schema:SoftwareApplication*). As shown in Figure 1, we extend Schema.org to align our concept to that vocabulary and inherit all its metadata properties. An advantage of using Codemeta is that it has become a reference vocabulary between different code repositories, and hence it facilitates interoperability between software metadata entries.

We have also extended the W3C Data Cubes standard [26] with dataset specifications (subclass of *qb:DataStructureDefinition*, as indicated in Figure 1). The Data Cube standard defines how to structure n-dimensional cubes of observations, and although it is tailored towards representing the observed values, the representation of the structure of a cube satisfies our needs for dataset representation.

Next, we reused NASA's Quantities, Units, Dimensions, and Data Types vocabulary(QUDT)[19] for representing units of variables, enriched with the canonicalization compound unit representation and transformation ontology (CCUT)[20] that provides additional properties to describe how to perform unit transformations.

Finally, we extend the DockerPedia ontology [14] to describe the executable containers that may be associated with a software component. DockerPedia was initially aimed at representing Docker containers, but other container frameworks can be easily represented in a similar manner. DockerPedia includes classes and properties to represent very specific descriptions of a software container, such as the software packages included, size of the image, dependent layers, testing commands, etc. Having these descriptions is useful to find commonalities between different software and potential vulnerabilities.

2) Overview of the Software Description Ontology

Figure 1 shows an overview of the main concepts of the Software Description Ontology. We use *sd:Software* as a general concept that refers to any piece of software we may want to describe. Scientific software is quite diverse, and may include generic software packages such as Scikit-learn[21] (a popular machine learning analysis framework), web services or concrete software scripts configured to run with a particular dataset. Software may have one or more *sd:SoftwareVersions*, which represent the evolution of a software component across time. It is important to describe the code for different software versions separately when reporting scientific results, as otherwise the results may differ from previous executions with another software version. Versions may be associated with one or more *sd:SoftwareConfigurations*, which represent a unique executable function of a particular software. For example, one software configuration of Scikit-learn may include a primitive invocation that imputes a target dataset, while another one may expose a trained model with a classifier. Software configurations are key to appropriately capturing heterogeneous functions in complex software libraries that can be used for multiple purposes (e.g., dataset analysis, creating visualizations of results, preparing data, etc.); and capture how a software component is invoked.

[18] https://www.earthsystemcog.org/projects/esmf/
[19] http://www.qudt.org/release2/qudt-catalog.html
[20] https://w3id.org/mint/ccut#
[21] https://scikit-learn.org/

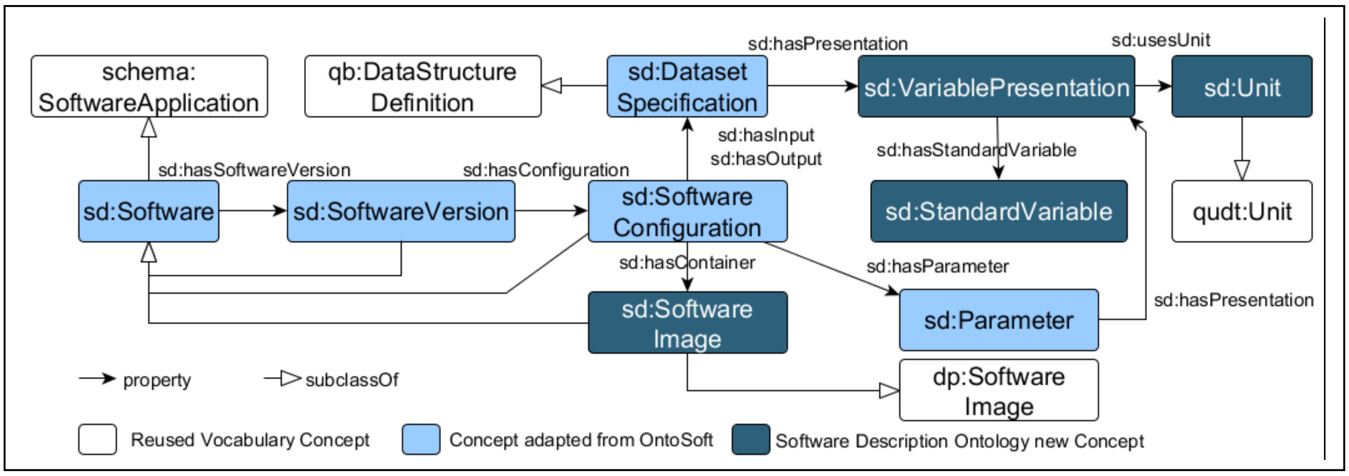


Figure 1: Overview of the main concepts of the Software Description Ontology, used to model OKG-Soft.

Software configurations also link to the expected structure (*sd:DatasetSpecification)* of those inputs (*sd:hasInput*), parameters (*sd:Parameter*) and outputs (*sd:hasOutput*) used or produced by software. Note that dataset specifications may define the structure of other sources besides files, such as data streams, APIs or database accesses that software connects to. Dataset specifications capture critical metadata such as the format of an expected input (*sd:hasFormat*) and the variables it may contain (*sd:VariablePresentation*). Examples of variables include the population of a district in a city from a census file used for creating a map visualization; precipitation measurements used to create a weather report; or the prediction of a trained machine learning model. Most software use a structured format to read data from inputs and serialize the output, even when performing simple tasks such as split and merge data. By capturing the expected structure of inputs and outputs, we can relate different software configurations in terms of the expected variables they use and produce. Software using generic functions on variables (e.g., calculating the average of a column in a file) may be described using anonymous variables.

Different software configurations may use different identifiers to refer to their variables. For instance, a climate model may expect a CSV file with a variable named "PREC" to refer to precipitation. Other software may use "P" to refer to the same variable. Fortunately, different scientific communities have developed and adopted naming standards, such as Climate and Forecast (CF)[22] in the climate community or the Scientific Variables Ontology[23] in geosciences, which aim at helping scientists use the same variable names based on their meaning. We have included the term *sd:StandardVariable* to allow linking different variables in dataset specifications to an existing standard (such as CF).

3) Accessibility and Modularity

Each version of the Software Description Ontology (SD) is stored independently and can be found in human readable and machine readable way to facilitate its reusability. The ontology aims to describe common aspects of software, and is organized in a modular manner. Therefore, anyone can import our ontology as part of another ontology that describes software at a more granular level. As an example, in our work we extended SD in the Software Description Ontology for Models (https://w3id.org/okn/o/sdm), which contains properties and classes specific to software for scientific models (e.g., spatial grids, time intervals at which the model operates, model assumptions, equations, processes captured by a model, etc.)

B. Publishing Software Metadata in the Web of Data

A major aspect of how OKG-Soft supports FAIR principles for software is the publication of software descriptions as an open knowledge graph in the Web of Data according the Linked Data principles [5], [6]: 1) we used derreferenceable HTTP URIs as identifiers for all the elements in the graph; 2) we used W3C standards (RDF [27] and SPARQL [28]) to return valuable information when accessing a URI, and 3) we linked relevant URIs together. We also used a permanent URI structure to ensure the long term availability of URIs, following the convention:

https://w3id.org/okn/i/[datasetID]/[instanceName]

Where *datasetID* denotes the name of the dataset we want to contribute to (e.g., in case we want to organize the graph for different software communities) and *instanceName* represents the identifier of a resource in the dataset.

Next, we populated OKG-Soft in two phases. In the first phase, we conducted a manual collection process of the information about the software models we wanted to add to the graph. In the second phase we expanded the collected information by linking it to external knowledge graphs with additional metadata. We describe these phases below.

1) Manual Software Metadata Collection

In a first iteration, we held a series of community meetings and workshops with software developers and environmental modelers to collect the metadata needed to describe and execute complex environmental models, including their data preparation and post-processing steps. This effort was performed within the scope of the Model Integration project (MINT) [29], which aims to provide a framework to reduce the time needed to integrate and compose complex software models from different disciplines, ranging from Climate to Agriculture or Economy. These types of software models are usually complex to set up and prepare data for, and therefore constitute an excellent testing ground for our ontology and knowledge graph. As a result of this process, we added 8 models (2 from climate sciences, 4 from hydrology, 2 from

[22] http://cfconventions.org

[23] http://www.geoscienceontology.org/svo

agriculture and 1 from economics), 31 software configurations that include software on how to transform, prepare and visualize data for the included models; and more than 280 relevant variables which describe their input and output data in detail.

2) *Linking OKG-Soft to the Web of Data*

We have enriched OKG-Soft by linking parts of the graph to existing initiatives and knowledge graphs for describing different aspects of software. We describe them below:

a) Semantic Description of Units.

Expert users and developers describe units associated with variables in a human readable manner (e.g., “m/day”). However, this representation is not enough if we aim to automatically find compatible variables (authors may describe variables using different notations) or plan to perform automated unit transformations. We need a semantic representation of units and their atomic components so they can be effectively reused by automated systems.

We leverage the work presented in [30] to transform the textual representations into a structured format. After this transformation, a unit such as “m/day” would turn into the structured RDF representation presented in Listing 1. The listing includes prefixes to represent the namespace of the vocabularies that are used, i.e., qudt (the NASA standard), ccut (a custom extension of qudt presented in [30]), mint (to refer to the entities in OKG-Soft) and rdfs (W3C standard). This representation has a clear separation between a target unit (m/day) its constituents (meter, day), their dimensions (length, time) and their relationship (length by time). Other units such as “km/month” share the dimension “L T-1” (length by time), making it easy to assess their compatibility.

b) Semantic Description of Variables

We use the Scientific Variables Ontology (SVO) [31],[24] an evolved version of the Geoscience Standard Names [32], as our main vocabulary for linking variables to a unique unambiguous standard representation. SVO was designed to serve as a semantic mediation hub between software models and it defines a set of principles and guidelines to create unique variable identifiers based on its characteristics. For example, while two different software models may refer to “temperature” as a variable in their inputs, the first model may expect temperature to be at sea level, while the second model may expect it at a certain soil reference depth. In SVO, both of these variables correspond to two separate identifiers, namely “sea_surface_water__temperature” and “soil__reference_depth_temperature”, which are semantically related to “temperature”.

We used the identifiers provided by expert modelers and software developers (built following the principles from SVO) to extract the context associated with each identifier and bring it into OKG-Soft. An example can be seen in Listing 2 for a variable “PRCP” that belongs to a hydrology model and corresponds to “main__precipitation_leq_volume_flux”:

The example shows how one variable from a hydrology model (*pihm_PRCP*), defined as *PRCP* by the model in one of its input files, is associated with the unique standard variable *atmosphere_water__precipitation_leq_volume_flux*, which refers to *water*, quantifies a *precipitation* process and

```
 @prefix qudt: <http://qudt.org/1.1/schema/qudt# > .
 @prefix ccut: <https://www.w3id.org/mint/ccut#> .
 @prefix mint: < https://w3id.org/ okn/i/mint/> .
 @prefix rdfs:< http://www.w3.org/2000/01/rdf-schema# > .
mint:m_day_1L_T_1     a qudt:Unit;
   rdfs:label           "m day-1" ;
   ccut:hasPart        " mint: u_L_meter_m, mint: u__1_T_day_day ;
   ccut:hasDimension   "L T-1";
   qudt:abbreviation    "m day -1".
mint:u_L_meter_m      a qudt:Unit;
   rdfs:label                "m" ;
   ccut:hasDimension   "L";
   ccut:QuantityKind   qudt:Meter;
   ccut:symbol            "m".
mint:u__1_T_day_day a qudt:Unit;
   rdfs:label                 "day-1" ;
   ccut:exponent             "-1";
   ccut:hasDimension      "T";
   ccut:QuantityKind      qudt:Day;
   ccut:symbol        "d".
```

Listing 1: Machine readable representation of the unit “m/day”. The unit is divided in two parts (meter and day), also described as units.

```
 @prefix mint: < https://w3id.org/okn/i/mint/> .
 @prefix rdfs:<http://www.w3.org/2000/01/rdf-schema# > .
 @prefix sd: <https://w3id.org/okn/o/sd#> .
 @prefix svu: <http://www.geoscienceontology.org/svo/svu#> .
 mint:pihm_PRCP a sd:VariablePresentation;
    rdfs:label          "PRCP" ;
    dc:description “Value of precipitation“;
    sd:usesUnit ms:m_day_1L_T_1;
    sd:hasStandardVariable
 <main__precipitation_leq_volume_flux>.
 <main__precipitation_leq_volume_flux>
   rdfs:label "atmosphere_water__precipitation_leq_volume_flux";
   svu:describesProcess
 <http://www.geoscienceontology.org/svo/svl/process#precipitation> ;
    svu:hasObject
 <http://www.geoscienceontology.org/svo/svl/matter#water>,
 <http://www.geoscienceontology.org/svo/svl/body#atmosphere> ;
    svu:hasProperty
 <http://www.geoscienceontology.org/svo/svl/property#precipitation_l
 eq_volume_flux> ;
    svu:quantifiesProcess
 <http://www.geoscienceontology.org/svo/svl/process#precipitation> .
```

Listing 2: Machine readable description of a variable “PRCP” that is linked to SVO (main__precipitation_leq_volume_flux).

refers to the property *precipitation_leq_volume_flux*, a type of volumetric flux.

Variable context is crucial for proper interpretation and reusability of software. It can also be used to detect inconsistencies in the data, e.g., if the expected dimension linked by a standard variable does not match the variable presentation unit dimension.

c) Semantic Description of Software Images

Nowadays it is commonplace to use software containers to facilitate the execution and set up of software. In OKG-Soft any container framework can be linked. We are using Docker containers [25] that include all executable packages and dependencies required to run each of the software entries described in the knowledge graph. However, the content of a container may be difficult to explore by a user. We use DockerPedia’s web service[26] to analyze and create a semantic representation of each container. This analysis includes all the software dependencies of a software configuration. Listing 3 shows an excerpt of one of the container descriptions

[24] http://www.geoscienceontology.org/svo

[25] https://bit.ly/30EyFYS

[26] dockerpedia.inf.utfsm.cl/

```
@prefix dp:
<http://dockerpedia.inf.utfsm.cl/resource/SoftwareImage> .
@prefix dps:
<http://dockerpedia.inf.utfsm.cl/resource/PackageVersion> .
@prefix dpv: <http://dockerpedia.inf.utfsm.cl/vocab# >.
@prefix rdfs:< http://www.w3.org/2000/01/rdf-schema# > .
dp:mintproject-weather-generator_latest
   rdfs:label   "mintproject/weather-generator" ;
   dpv:imageIdentifier      "mintproject/weather-generator:latest" ;
   dpv:tag      "latest";
   dpv:containsSoftware dps:python-pip-9.0.1-2.3~ubuntu1,
dps:expat-2.2.5-3.
```

Listing 3: Fragment of a container description for a climate model

generated by DockerPedia for a climate model, which are also available as Web of Data objects (included software packages have been reduced for readability).

d) Expanding Software Descriptions with Wikidata

Wikidata[27] [33] is an open, crowdsourced knowledge base that contains more than 50 million statements about entities of interest in the world. Wikidata is a great source of machine-readable knowledge about entities relevant to software, and is continuously growing thanks to a community of users who curate available contents. We use Wikidata as an additional source to enrich software descriptions. We link software, atomic variables, and units that have an exact match in Wikidata to elements in OKG-Soft. We use an interactive process to clarify with a user whenever a term is ambiguous. For instance, terms like "albedo" may have several definitions in Wikidata, as it could be a physical property, a role-playing game or a color.

Listing 4 shows how an enriched term appears in OKG-Soft. Note the owl:SameAs link to the Wikidata term, which indicates that that the albedo entity in the software description and the Wikidata P4501 entity refer to the same thing. The *schema:description* definition is imported from Wikidata.

All the software developed to facilitate enriching and linking OKG-Soft with existing work is openly available online under a CC-BY license [34].

C. Programmatically Accessing OKG-Soft

In order to maximize the usability of the contents in OKG-Soft, we have strived to make the knowledge graph accessible to software developers and users with and without knowledge representation or RDF/SPARQL skills.

Figure 2 shows an overview of the OKG-Soft API architecture. The lower part of the figure depicts our SPARQL endpoint,[28] which we use to organize the contents of OKG-Soft according to the Software Description Ontology. The SPARQL endpoint is targeted towards users familiar with Semantic Web technologies. In order to manage software contributions from users, we have organized the SPARQL endpoint in named graphs [35], where each contributor can edit their own graph. Therefore, all users can explore software entries added by other users, but only corresponding authors can delete their own contents.

As shown in the medium part of Figure 2, we have designed REST APIs for developers without SPARQL skills to access, add and edit software entries in our SPARQL

```
@prefix sv:< http://www.geoscienceontology.org/svo/svl/property#> .
@prefix schema: <https://schema.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
sv:albedo
  rdfs:label "albedo","reflection_coefficient" ;
  owl:sameAs <http://www.wikidata.org/entity/P4501> ;
  schema:description "ratio of reflected radiation to incident radiation"
```

Listing 4: Linking OKG-Soft variables with Wikidata descriptions

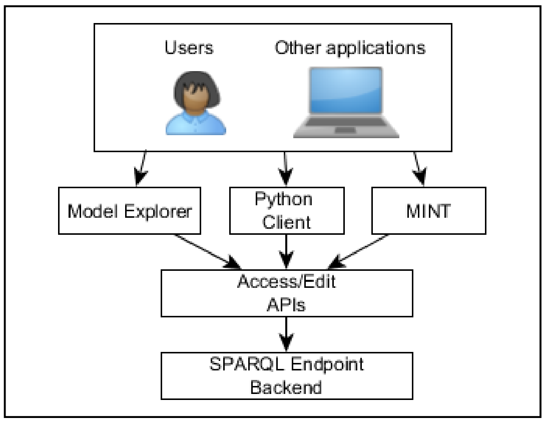


Figure 2 OKG-Soft API structure

endpoint. We have adopted GRLC [36] to implement our access queries, as it provides a framework to configure REST APIs by specifying SPARQL queries in a GitHub repository.[29] Whenever a new GET API call is required, we write the corresponding SPARQL query and GRCL will automatically make it available in the API. GRLC enables us to quickly integrate any new API requirements in a matter of seconds, without having to worry about configuration or deployment of the system.

New software entries for OKG-Soft are validated and managed through an Open API implementation,[30] following the best practices adopted by developers. Developers may issue requests in JSON or JSON-LD [37], following the classes and properties defined in our ontology. We do not require creating complete software entries. Instead, software developers may edit and expand existing software entries by using a PUT operation. Since the amount of triples required to fully describe a software component is usually within hundreds of triples, we do not expect scalability issues (current triple stores can handle millions of triples without an issue). Both access and edit APIs can be found online[31] along with documentation and examples.

Finally, as shown in the center of Figure 2, we have also designed programmatic clients to facilitate using our proposed APIs. In this case, our target are users who are familiar with scripting languages such as Python, but are not familiar with REST APIs. An example stating how to use our Python client can be found in an online notebook.[32]

IV. USING OKG-SOFT TO EXPLORE AND COMPOSE SOFTWARE

In order to illustrate the benefits of OKG-Soft, we show in this section how queries can be answered to obtain machine-readable metadata. We also show how it can be used to explore different software metadata at various levels of complexity through two applications that exploit the contents

[27] http://wikidata.org/
[28] https://endpoint.mint.isi.edu/ds/query
[29] https://github.com/mintproject/MINT-ModelCatalogQueries
[30] https://github.com/mintproject/MINT-ModelCatalogIngestionAPI
[31] https://query.mint.isi.edu/api/mintproject/MINT-ModelCatalogQueries#
[32] https://github.com/mintproject/MINT-ModelCatalogAPI-client

of the knowledge graph to facilitate software understanding and composition.

A. *Answering Queries about Software*

We designed queries to evaluate OKG-Soft based on the requirements collected in our ontology development phase. That is, the queries aim to test how well a software component can be described to ease its understanding (i.e., input and output description) and composition with other software. Given the domain we chose to populate the knowledge graph, we use queries based on environmental software models. However, these can be generalized to describe any other type of software. The queries and answers used in this section are available in [38].

Query 1: What is the basic description of a given software component?

This is perhaps the simplest query to start understanding the details of a software and its metadata, as it returns a description of its functionality, authorship and pointers to other details of software, such as its available versions, categories, etc. If we use Cycles[33] (an agriculture model derived from [39]) as our target software, the query would look as illustrated in Listing 5.

Query 2: What is the information about the execution requirements for all available versions and configurations of a given software component?

This query serves a dual purpose: it finds all the versions and configurations of a given software component and retrieves pointers to their executable information, i.e., the location of the scripts detailing how to invoke software and whether it has associated containers for its execution. For Cycles, the resultant query is shown in Listing 6.

Query 3: What are the expected inputs and outputs of a software configuration?

This query gives an insight into how a specific software component can be executed by listing its required inputs and expected outputs. Optional outputs are returned as part of a configuration, even if they are not always present in the execution of the component. In the case of Cycles, the query in Listing 7 returns all its configurations, with their associated inputs and outputs grouped together.

Query 4: Given an input or output, what are its associated variables and metadata?

Once the inputs and outputs used by a software configuration are clear, the next step is to find more about their contents, i.e., the different variables they describe. For example, in the previous example one of the inputs of Cycles is a "weather" file (*mint:cycles_weather*), that contains relevant variables for the software model related to weather. Listing 8 displays the query required to further describe the file. The results of the query produce a table, where each row is a variable with its label, its description, the units it is measured in and the standards variable it corresponds to. For example, one of the variables included in the file is "Tn" (minimum temperature of the day), measured in Celsius, which corresponds to the SVO term "air__daily_min_of_temperature".

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?property ?value
WHERE {
  ?software ?property ?value.
  ?software rdfs:label "Cycles"
}
```

Listing 5: Query to retrieve basic descriptions of software

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sd: <https://w3id.org/okn/o/sd#>
select  ?version ?configuration where {
 ?model sd:hasSoftwareVersion ?version;
    rdfs:label "Cycles".
 ?version sd:hasConfiguration ?configuration.
 OPTIONAL {?configuration sd:hasComponentLocation ?loc}
 OPTIONAL {?configuration sd:hasContainer ?cont.}
```

Listing 6: Query to retrieve all versions and configurations of a software component (?soft) and their respective executable information.

```
@ prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@ prefix sd: <https://w3id.org/okn/o/sd#> .
select   distinct  ?configuration  (group_concat  (distinct  ?input;
separator=', ') as ?input_variables) ((group_concat (distinct ?output;
separator =', ') as ?output_variables) where {
 ?soft sd:hasSoftwareVersion ?version.
 ?soft rdfs:label "Cycles".
 ?version sd:hasConfiguration ?configuration.
 ?configuration sd:hasInput/rdfs:label ?input;
                    sd:hasOutput/rdfs:label ?output .
}
```

Listing 7: Query to retrieve inputs and output of a software component.I

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sd: <https://w3id.org/okn/o/sd#>
select distinct ?model_config where {
  ?io a sd:DatasetSpecification.
  ?io   sd:hasPresentation   /   sd:hasStandardVariable   /   rdfs:label
"air__daily_min_of_temperature".
  ?soft_config sd:hasOutput ?io.
}
```

Listing 8: Query to retrieve the inputs and outputs of a software configuration (?soft_config)

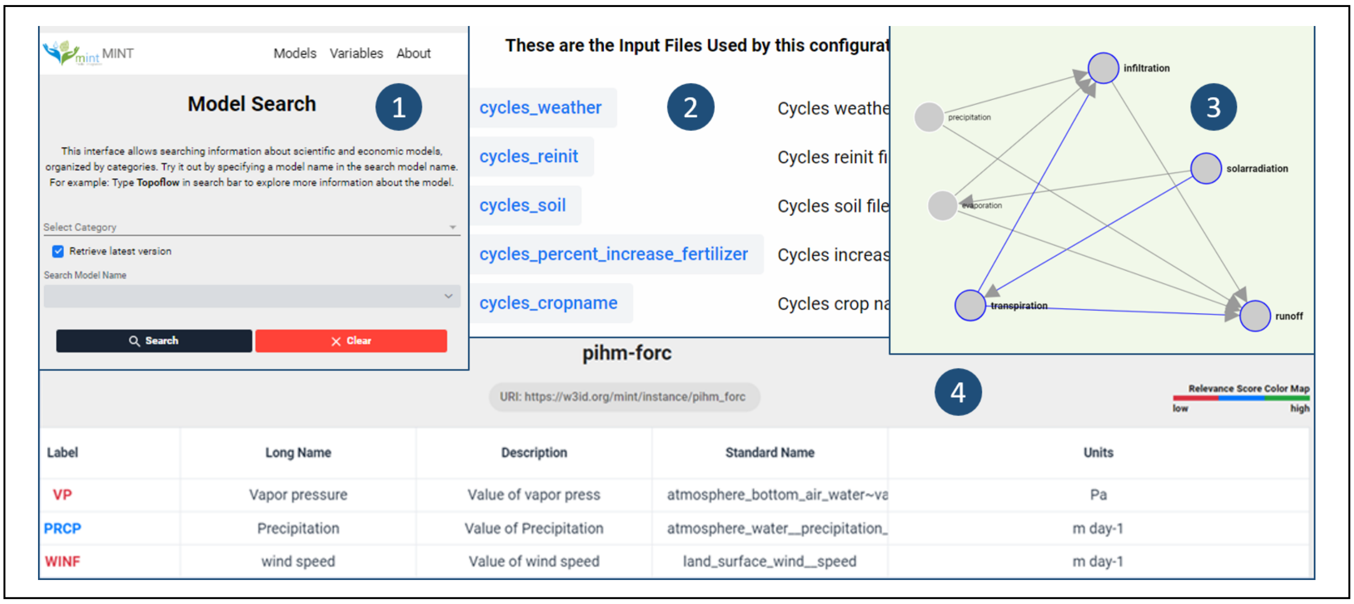
```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix sd: <https://w3id.org/okn/o/sd#>
prefix mint: < https://w3id.org/okn/i/mint />
select  distinct  ?label ?longName ?unit ?sn where {
mint:cycles_weather sd:hasPresentation ?variable.
    ?variable sd:usesUnit/rdfs:label ?unit;
         rdfs:label ?label;
         sd:hasLongName ?longName;
         sd:hasStandardVariable/rdfs:label ?sn.
}
```

Listing 9: Query to retrieve software compatible with a given software input (mint:cycles_weather)

Query 5: Which software produces a variable that may be used as input for another software component?

This query expands on how a software component may be composed with other software. Specifically, it retrieves those software entries that produce a resulting variable required to run another software. Following our previous example for

[33]https://plantscience.psu.edu/research/labs/kemanian/models-and-tools/cycles



Model Search

This interface allows searching information about scientific and economic models, organized by categories. Try it out by specifying a model name in the search model name. For example: Type **Topoflow** in search bar to explore more information about the model.

☑ Retrieve latest version

These are the Input Files Used by this configurat

cycles_weather	Cycles weathe
cycles_reinit	Cycles reinit fi
cycles_soil	Cycles soil file
cycles_percent_increase_fertilizer	Cycles increas
cycles_cropname	Cycles crop na

pihm-forc

URI: https://w3id.org/mint/instance/pihm_forc

Label	Long Name	Description	Standard Name	Units
VP	Vapor pressure	Value of vapor press	atmosphere_bottom_air_water~va	Pa
PRCP	Precipitation	Value of Precipitation	atmosphere_water__precipitation_	m day-1
WINF	wind speed	Value of wind speed	land_surface_wind__speed	m day-1

Figure 3: An overview of the Model Explorer, and application to navigate scientific software models included in OKG-Soft. Model Explorer allows searching for models (1), analyzing which are the inputs and outputs of a given model configuration (2) viewing potential relationships between variables (3) and finding out more about the contents of model files (4)

Cycles, we can query for software entries that generate estimates on the minimum temperature of the day, as shown in Listing 9.

The result of the query (available online with the rest of the queries shown in this section [38]) returns a software configuration from a weather generator which can be used to inform an agriculture model.

A very similar query (by replacing *sd:hasOutput* with *sd:hasInput*) may be used to retrieve which software configurations are compatible with a given result from Cycles, and thus retrieve which other software entries Cycles can be combined with.

In summary, these queries show how the contents of OKG-Soft may be used to gather insight about the different ways of running software, its expected inputs, outputs and variables; and how it may interoperate with other software. All queries return results in less than a second.

B. *An Application for Exploring Software Components: Model Explorer*

We have designed Model Explorer,[34] an application for finding and exploring software models and metadata available in OKG-Soft without having to interact with the APIs or clients designed for developers. A snapshot of the application can be seen in Figure 3, highlighting the main capabilities of Model Explorer.

As shown on the top left of Figure 3, users can search for existing software models by typing their names, or sorting them out by category (e.g., Agriculture, Climate, etc.). Once a model is selected, the application will show its available versions and software configurations so users can explore their corresponding inputs and outputs (highlight 2 on Figure 3). If several software configurations are available for a particular model version, the application will display them side by side to enable comparison. If several versions of a software component are available, by default the system will display the last one, allowing users to select others.

Elements shown in model configurations are interactive and will lead the users to pages with more information about a software component. For example, highlight 4 in Figure 3 shows the variable description table that appears after clicking on one of the inputs of a hydrology model. Highlight 3 shows an interactive graph of the variables included on a model, and how they affect each other in a particular software configuration. The code of the Model Explorer application is available online.[35]

C. *An Application for Facilitating the Integration and Composition of Software Models: MINT*

One of the most challenging aspects of using software models is selecting the appropriate input data products and preparing them according to the models to run. MINT [29], [40] is a novel framework for model integration that uses semantic representations to describe datasets and models to support users in data search and transformation; model selection, setup and combination into end-to-end workflows, and execution and visualization of results. MINT is integrated with OKG-Soft, exploiting the contents of the knowledge graph to assist users when running individual software models or when running software models in combination with other models. Figure 4 shows a simplified overview of how the system interacts with users to assist in their analysis. First, users specify a set of variables of interest for the modeling question they want to address. For example, in Figure 4 a user is interested in analyzing how different aspects of rainfall would affect crop production for a given region. MINT uses this information to locate software models that produce the target variable (crop production), and how to derive it from the driving variable of the analysis (rainfall). Since rainfall is a variable that could refer to different specific properties (mass flux, average volume, volume flux or time integral of volume flux), the system considers all of them when looking for candidate software models. Then, MINT suggests appropriate

[34] http://models.mint.isi.edu

[35] https://github.com/mintproject/MINT-ModelCatalogExplorer/

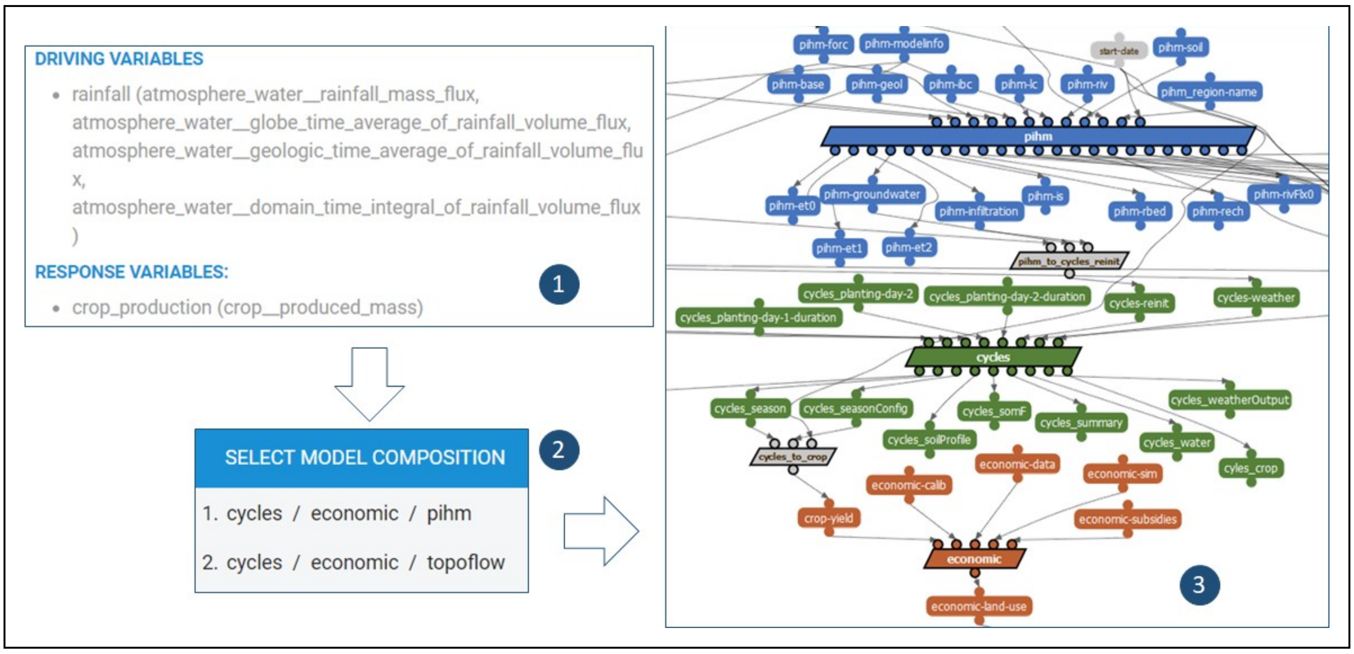


Figure 4: Overview of the process followed by MINT to guide users for model composition. First, users select the driving variables (inputs) and response variables (outputs) to focus on (1). Then, the system looks for available combinations of software models from OKG-Soft (2). Finally, a workflow including all necessary transformations is shown to users (3), highlighting different modeling domains in different colors.

combinations of models, along with the necessary data transformations composed together as a workflow.

V. Conclusions and Future Work

In this paper we have presented OKG-Soft, a framework for capturing, publishing, and using machine-readable metadata for software. We have organized the contents of OKG-Soft by extending the Software Description Ontology with descriptions of input and output datasets of the software in terms of format and contents. We have tested our ontology by modeling and curating environmental software, including domains such as hydrology, climate and agriculture. We publish the software descriptions as an open knowledge graph with links to the Web of Data using Linked Data principles. Our framework supports the FAIR principles with respect to scientific software by exposing software descriptions on public APIs for search purposes (findability), using persistent HTTP URIs to refer to software and its associated contents (accessibility), using common vocabularies and standards to describe software (interoperability) and by describing the inputs and outputs of software (easing reusability). Two applications (Model Explorer and MINT) demonstrate how the infrastructure we developed may be used to explore and combine software models.

We are supporting additional queries and improving the documentation of our APIs to facilitate their usability. We are also working with third party developers of scientific software who are starting to and curate the contents of OKG-Soft and posing new requirements.

As for the population of OKG-Soft, while adding a new entry on the knowledge graph can be achieved through an API call, we still have to address the curation and editing of software metadata. Creating complete metadata for a piece of scientific software can take a significant amount of time for contributors, but we argue that the process only needs to be completed once, and the payoff in terms of composition, understanding and reusability is worth the extra effort. Nevertheless, our current and future work aims to address this issue by moving towards an automatic metadata ingestion approach. We plan on testing text metadata extraction techniques to retrieve relevant software metadata (license, contributors, variable information, etc.) typically buried in readme files, instruction manuals or even source code.

We are also working on describing the format of files in terms of how variables are represented within input and output files (e.g., their position in a CSV), so as to automatically reformat files and convert variables. Finally, we are working towards creating an executable environment to test the software entries defined within OKG-Soft.

Acknowledgments

This work was funded by the Defense Advanced Research Projects Agency with award W911NF-18-1-0027 and the National Science Foundation with award ICER-1440323. We would like to thank Dhruv Pattel, Rohit Mayura, Amrish Goel and Anuj Doiphode for their help and work in the Model Explorer and APIs. We would also like to thank all the researchers who have contributed to OKG-Soft and MINT, including Christopher Duffy, Lele Shu, Scott Peckham, Armen, Kemanian, Kelly Cobourn, Zeya Zhang and Maria Stoica.

References

[1] Y. Gil *et al.*, “Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance: Geoscience Paper of the Future,” *Earth Space Sci.*, vol. 3, no. 10, pp. 388–415, Oct. 2016.

[2] D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble, “Common motifs in scientific workflows: An empirical analysis,” *Future Gener. Comput. Syst.*, vol. 36, pp. 338–351, 2014.

[3] Y. Gil, V. Ratnakar, and D. Garijo, “OntoSoft: Capturing scientific software metadata,” in *Proceedings of the 8th International Conference on Knowledge Capture*, 2015, p. 32.

[4] L. Carvalho, D. Garijo, C. B. Medeiros, and Y. Gil, “Semantic Software Metadata for Workflow Exploration and Evolution,”

in *Proceedings of the Fourteenth IEEE International Conference on eScience*, Amsterdam, The Netherlands, 2018.

[5] T. Berners-Lee, "Linked Data: Design Issues," World Wide Web Consortium, Jul. 2006.

[6] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, 2009.

[7] M. D. Wilkinson *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Sci. Data*, vol. 3, p. 160018, Mar. 2016.

[8] Y. Gil, V. Ratnakar, and D. Garijo, "OntoSoft: Capturing Scientific Software Metadata," 2015, pp. 1–4.

[9] M. B. Jones *et al.*, "CodeMeta: an exchange schema for software metadata. KNB Data Repository." KNB Data Repository, 2016.

[10] D. Oberle, S. Lamparter, S. Grimm, D. Vrandečić, S. Staab, and A. Gangemi, "Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems," *Appl Ontol*, vol. 1, no. 2, pp. 163–202, Apr. 2006.

[11] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider, "Sweetening Ontologies with DOLCE," in *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, Berlin, Heidelberg, 2002, pp. 166–181.

[12] A. D. Spear, W. Ceusters, and B. Smith, "Functions in Basic Formal Ontology," *Appl. Ontol.*, vol. 11, no. 2, pp. 103–128, Jun. 2016.

[13] J. Ison *et al.*, "EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats," *Bioinformatics*, vol. 29, no. 10, pp. 1325–1332, May 2013.

[14] M. Osorio, H. Vargas, and C. Buil Aranda, "DockerPedia: a Knowledge Graph of Docker Images," in *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018)*, Monterrey, 2018.

[15] I. Santana-Peréz and M. Pérez-Hernández, "Towards Reproducibility in Scientific Workflows: An Infrastructure-Based Approach," *Sci. Program.*, vol. 2015, p. 11, 2015.

[16] S. D. Peckham, E. W. H. Hutton, and B. Norris, "A component-based approach to integrated modeling in the geosciences: The design of CSDMS," *Comput. Geosci.*, vol. 53, pp. 3–12, 2013.

[17] L. Shamir *et al.*, "Practices in source code sharing in astrophysics," *Astron. Comput.*, vol. 1, pp. 54–58, Feb. 2013.

[18] Y. Gil, D. Garijo, S. Mishra, and V. Ratnakar, "OntoSoft: A distributed semantic registry for scientific software," in *e-Science (e-Science), 2016 IEEE 12th International Conference on*, 2016, pp. 331–336.

[19] L. Zentner, M. Zentner, V. Farnsworth, M. McLennan, K. Madhavan, and G. Klimeck, "nanoHUB.org: Experiences and Challenges in Software Sustainability for a Large Scientific Community," *J. Open Res. Softw.*, vol. 2, no. 1, Jul. 2014.

[20] I. D. Dinov *et al.*, "Efficient, Distributed and Interactive Neuroimaging Data Analysis Using the LONI Pipeline," in *Frontiers in Neuroinformatics*, 2009, vol. 3.

[21] B. Giardine *et al.*, "Galaxy: a platform for interactive large-scale genome analysis," *Genome Res.*, vol. 15, no. 10, pp. 1451–1455, Oct. 2005.

[22] K. Wolstencroft *et al.*, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Res.*, 2013.

[23] Y. Gil, "Workflow Composition: Semantic Representations for Flexible Automation," in *Workflows for e-Science*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds. London: Springer London, 2007, pp. 244–257.

[24] A.-L. Lamprecht, S. Naujokat, B. Steffen, and T. Margaria, "Constraint-Guided Workflow Composition Based on the EDAM Ontology," *CoRR*, vol. abs/1012.1640, 2010.

[25] D. Garijo and D. Khider, "Requirements for the Software Description Ontology (May, 2019)." Figshare, 2019.

[26] R. Cyganiak and D. Reynolds, "The RDF Data Cube Vocabulary," W3C, W3C Recommendation, Jan. 2014.

[27] G. Klyne, J. J. Carroll, and B. McBride, *Resource Description Framework (RDF): Concepts and Abstract Syntax*. 2004.

[28] S. Harris and A. Seaborne, "SPARQL 1.1 Query Language," W3C, W3C Recommendation, Mar. 2013.

[29] Y. Gil *et al.*, "MINT: Model Integration Through Knowledge-Powered Data and Process Composition," in *Proceedings of the Ninth International Congress on Environmental Modeling and Software*, Ft Collins, CO, 2018.

[30] B. Shbita, A. Rajendran, P. Jay, and K. Craig, "Parsing, Representing and Transforming Units of Measure," presented at the Modeling the World's Systems, 2019.

[31] M. Stoica and S. D. Peckham, "An Ontology Blueprint for Constructing Qualitative and Quantitative Scientific Variables," in *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018.*, 2018.

[32] S. D. Peckham, E. W. H. Hutton, and B. Norris, "A component-based approach to integrated modeling in the geosciences: The design of CSDMS," *Comput. Geosci.*, vol. 53, pp. 3–12, 2013.

[33] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Sep. 2014.

[34] D. Garijo, D. Khider, and Y. Gil, "OKG-Soft code release (1.0.0)." Zenodo, 02-May-2019.

[35] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler, "Named graphs, provenance and trust," in *Proceedings of the 14th international conference on World Wide Web - WWW '05*, Chiba, Japan, 2005, p. 613.

[36] A. Meroño-Peñuela and R. Hoekstra, "GRLC Makes GitHub Taste Like Linked Data APIs," in *The Semantic Web*, vol. 9989, H. Sack, G. Rizzo, N. Steinmetz, D. Mladenić, S. Auer, and C. Lange, Eds. Cham: Springer International Publishing, 2016, pp. 342–353.

[37] G. Kellogg, M. Lanthaler, and M. Sporny, "JSON-LD 1.0," W3C, W3C Recommendation, Jan. 2014.

[38] D. Garijo, "Validation queries for the paper OKG-Soft: An Open Knowledge Graph with Machine Readable Scientific Software Metadata." Figshare, 2019.

[39] A. R. Kemanian and C. O. Stöckle, "C-Farm: A simple model to evaluate the carbon balance of soil profiles," *Eur. J. Agron.*, vol. 32, no. 1, pp. 22–29, Jan. 2010.

[40] D. Garijo *et al.*, "An Intelligent Interface for Integrating Climate, Hydrology, Agriculture, and Socioeconomic Models," in *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion*, New York, NY, USA, 2019, pp. 111–112.