Semantic Software Metadata for Workflow Exploration and Evolution

Lucas A. M. C. Carvalho
Institute of Computing
University of Campinas
Campinas, SP, Brazil
lucas.carvalho@ic.unicamp.br

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

Claudia Bauzer Medeiros
Institute of Computing
University of Campinas
Campinas, SP, Brazil
cmbm@ic.unicamp.br

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

***Abstract* − Scientific workflow management systems play a major role in the design, execution and documentation of computational experiments. However, they have limited support for managing workflow evolution and exploration because they lack rich metadata for the software that implements workflow components. Such metadata could be used to support scientists in exploring local adjustments to a workflow, replacing components with similar software, or upgrading components upon release of newer software versions. To address this challenge, we propose OntoSoft-VFF (Ontology for Software Version, Function and Functionality), a software metadata repository designed to capture information about software and workflow components that is important for managing workflow exploration and evolution. Our approach uses a novel ontology to describe the functionality and evolution through time of any software used to create workflow components. OntoSoft-VFF is implemented as an online catalog that stores semantic metadata for software to enable workflow exploration through understanding of software functionality and evolution. The catalog also supports comparison and semantic search of software metadata. We showcase OntoSoft-VFF using machine learning workflow examples. We validate our approach by testing that a workflow system could compare differences in software metadata, explain software updates and describe the general functionality of workflow steps.**

Keywords—scientific workflows; software metadata; software functions; software registries; workflow evolution.

I. INTRODUCTION

Workflow management systems [1] play a major role in supporting scientists to design, document and execute their computational experiments. During workflow design, scientists use third party software or their own code to implement workflow components. This paper investigates the issues that arise when such software evolves in terms of how a scientist's workflow is affected.

There are many reasons for scientists to modify a workflow that they created, either by changing specific steps of the workflow (also called *workflow components*) or changing the workflow structure. Changes in software used to implement components are common and could happen for different reasons, e.g., a newer version is available, older software is not maintained. Also, data sources change, e.g. when datasets are updated with new formats, which may require adjustments in existing components and adding new ones. Thus, due to changes in software and data, workflows must be updated accordingly to avoid workflow decay [2] and reproducibility issues [13]. Another important reason to update workflows is when scientists are exploring alternative ways of performing a computational experiment. During these exploratory tasks, scientists often want to compare methods or try different approaches to implement a workflow component.

In current workflow systems, scientists manage these updates manually. However, updating a workflow is a complex and time-consuming task, as it requires tracking down information about the different versions of software and functions used in the components of the workflow and understanding the impact in other workflow steps.

In previous work [4], we elicited a set of requirements for supporting the exploration and update of workflows motivated by hydrology workflows and their use of models with very different versions over the years. These requirements motivate the need for capturing additional metadata for better describing software used in workflows, such as software functionality and implementation changes over time. This paper revisits those requirements and makes them more specific through a detailed scenario in which a decades-old machine learning software is used in a workflow for weather prediction. Those requirements guided the design and implementation of OntoSoft-VFF, a

Proceedings of the 14th IEEE International Conference on eScience, Amsterdam, The Netherlands, October 2018.

semantic software metadata catalog to help scientists to manage workflow evolution and updates. OntoSoft-VFF is based on a novel ontology for representing software metadata. Our goal is to use the information in OntoSoft-VFF to support scientists in selecting appropriate pieces of software to implement a given workflow component, to explore the use of alternative software in their workflow, and to keep track of all workflow changes. OntoSoft-VFF's catalog extends OntoSoft [5], an existing metadata catalog designed for fostering scientific software reuse and sharing.

The main contributions of this paper are the following:

- Through a scenario in which a scientist updates a computational experiment, we revisit requirements for software metadata to describe software that implements workflow components.
- A software metadata catalog developed for those requirements. The catalog is based on a novel ontology designed to describe software functionality and its evolution. The catalog supports comparing and searching semantic metadata for software.

To illustrate our work, we use a running example of machine learning workflows, since it is a domain with many alternative methods available where software changes frequently. This example is used to present the elements of our ontology and the features available in the catalog. We show how to create and update workflow components using the semantic metadata stored in our catalog, and the benefits of integrating the metadata catalog with a workflow system to support scientists in their exploratory tasks. Finally, we validated our approach showing how it addresses our requirements. OntoSoft-VFF has been designed to be generic and can be applied to any scientific domain.

Throughout this paper we adopt the following terminology:

- *Software* is a set of functions that perform similar or related computations and are delivered as a package by developers. An example of software is Weka [7], a decades-old open source Java software with a widely used collection of machine learning algorithms for data mining tasks. A more modern and also popular software is the Scikit-learn Python libraries [17].
- *Software version* is a unique state of a software as it is being released. For example, the latest Weka release is version 3.9.2.
- *Functionality* is a conceptual computation or operation that can be performed by a piece of software. For example, Weka implements classification, regression, and clustering functionalities.
- *Software function* is the implementation of a functionality in a software. An example is the Weka J48 function that implements a classification functionality using the C4.5 decision tree algorithm [18].
- *Software change* is a relevant modification associated with a software function over time. An example of a change is the improvement of accuracy in the result of a J48 classifier function.

A new software version may imply software changes as well as modified, new, or deprecated functionalities or functions.

The rest of the paper is organized as follows. Section II introduces related research on workflow updates and software representation. Section III describes the main scenario we are addressing, expanding the requirements derived from previous work. Section IV introduces OntoSoft-VFF. Section V shows how we exploit the data published in the catalog to facilitate workflow exploration and evolution. In Section VI we validate our framework against the requirements in Section III. Finally, we present our conclusions and future work in Section VII.

II. Related Work

Our discussion of related work covers two areas: approaches for workflow exploration and updates, and approaches for software representations and software changes.

Workflow systems [1, 14, 15, 16] are mostly concerned with workflow construction, execution and provenance collection and inspection. Vistrails [3, 16] uses a software registry that stores the software name and version identification to support workflow upgrades. However, this registry does not track changes in terms of functions and functionality, nor store information about semantics of inputs and outputs, such as data types and data formats. Such kinds of metadata are necessary to support more robust approaches to update a workflow with components for data transformation and other upgrades.

In [12], the authors proposed a framework for management of knowledge associated with workflow evolution. The framework, however, does not track changes in the software used to implement the workflow components, thus missing the opportunity to relate the effects of changes in software to the outputs of workflow components.

Understanding how results have been produced requires knowledge of the software being used. Work such as [6,8,11] proposes mechanisms to represent software; however, they lack metadata for describing changes in software, and how to use specific software functionalities. Their representations define inputs and outputs for the software in general, rather than defining the inputs and outputs for a function. OntoSoft [6] is a software registry that is concerned with representation, sharing and reuse of software metadata. The software representation used does not capture information about software functions and software changes over time.

Regarding software updates, software version management systems [11] track changes in software code. However, changes represented in these systems do not provide enough information to allow a scientist to filter them by software function, for example, and track down the changes through time associated with a specific function or functionality. Also, it is hard for a scientist to track the issues and bug fixes related to a specific function in a software version, because version control systems do not explicit represent the functions available in the versioned software.

The most common shortcoming of these approaches is the lack of appropriate metadata to account for and appropriately track software evolution, thereby placing a major burden to scientists in managing workflow evolution. Even when metadata exists, it is not designed to support workflow exploration and updates.

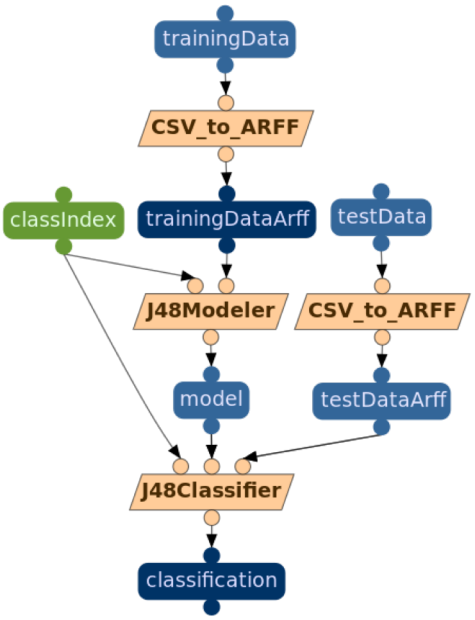

Figure 1. A very simple workflow using a decision tree machine learning algorithm for training and classification.

III. Motivating Scenario – Revisiting Requirements for Workflow Exploration and Updates

In this section we show how scientists explore new functionalities of software, so they can upgrade workflow components using these new functionalities. In order to achieve this goal, scientists usually go through a series of tasks that are performed manually and without any support. They start by identifying the function and software used in a workflow component, and understanding the algorithms that they implement. Scientists also need to find and compare similar software versions and functions. Finally, when deciding either which workflow component to upgrade or which software function to use to implement a new component, scientists need to be able to create or upgrade components and to change the rest of the workflow as needed.

To illustrate the needs for supporting the management of workflow exploration and evolution, we use an example of a workflow designed to process weather data to make weather predictions. We chose this scenario for several reasons. First, it is a simple domain-independent scenario chosen to simplify our presentation, yet it captures the complexities that we have seen in our prior work with hydrology modeling software and workflows [4]. Scientific modeling software has the same issues but also additional subtleties as discussed in [4]. Second, the scenario involves machine learning algorithms, and consequently a large choice of options of algorithms (and thus many exploration and update options). Lastly, we had implemented a variety of workflows in previous work (e.g., [6;19;20]) using older versions of Weka, as well as new ones using Scikit-learn and other machine learning libraries [21]. The workflows run in the WINGS [1] workflow system.

In our scenario, Alice, a meteorologist in California, wants to predict weather for the city of Pasadena. She starts with a very simple workflow, shown in Figure 1, that had been used to process 2007 weather data from Santa Monica to make weather predictions for Pasadena, both cities located in California. The workflow no longer runs, and Alice would like to update it.

The workflow contains two workflow components (*J48Modeler* and *J48Classifier*) from Weka that use the C4.5 decision tree algorithm. The first component uses it to learn a decision tree model from training data (the *trainingData* input), while the *J48Classifier* uses this learned model to classify test data (the *testData* input). The *ClassIndex* parameter, used as input for both workflow components, specifies that the feature in a specific indexed column is the one we are trying to predict. The workflow also contains two components to ingest data since Weka uses a special comma separated data format called ARFF (Attribute-Relation File Format).

In the discussion that follows, we will describe how in order to decide whether (and how) to upgrade the workflow, Alice needs to go through the documentation provided by Weka in quite a bit of detail.

A. *Finding which software is used in workflow components*

A workflow does not include much information about the software that implements each workflow component, such as software function and version invoked. This information could help a scientist like Alice to upgrade the workflow component – e.g., to decide whether and how to upgrade to a new version.

Alice found in the workflow documentation that Weka is used to implement the workflow and that both workflow components invoke the Weka J48 functions. Alice has to read the Weka documentation to understand that these functions implement the C4.5 decision tree algorithm. To manage the workflow evolution, she needs to know not only the software and algorithm used to implement the workflow components, but also the specific software version and functionalities invoked in Weka to implement each of the workflow components. This information, however, is not explicitly captured by scientific workflow systems, which usually only store the code that invokes the software. So, either Alice relies on the documentation, or she looks at the actual component code to find out how the software has been invoked and which version is being used. The latter information, even if available, is not easy to infer from an invocation code.

In some cases, the function invocation can help to find out the code that is being used, but unfortunately this does not help a scientist to understand what the function does or the semantics of the inputs and outputs and of their data types or formats. This information is important for scientists to know what kind of data is needed to run a component as well as to check compatibility between workflow components.

Alice found out by looking at the code that both components are implemented using Weka version 3.6.2, which was released in 2010. Through further analysis, she discovers that the *J48Modeler* and *J48Classifier* components are implemented invoking the J48 Java class in Weka, which is located in the *weka.classifiers.trees* package and uses the C4.5 decision tree algorithm. Now, she is ready for the next step in deciding whether to upgrade the workflow – assessing the impact of changing software versions.

B. Understanding differences between software versions

Important changes may have been made between software version releases, such as the addition of new functions and changes in function interfaces which affect their invocation. To check whether and how to upgrade a workflow, scientists often read release notes where software developers usually describe differences between software versions, e.g., bug fixes or performance improvements. However, release notes may be generic and are not designed to allow scientists to quickly determine how general software changes affect their particular workflows.

Alice is more interested in stable versions than development versions, since the latter are more likely to suffer from bugs. However, stable versions are likely to suffer of delays in receiving new functionalities since they are tested in development versions first. Minor and patch versions, compared to major versions, usually provide backward compatibility changes that should not affect function interfaces, making workflow upgrades much easier. Changes in software interfaces may in turn affect the implementation of the interfaces of the workflow components. The solution is either to recode the component, or to create additional workflow components for data transformation to make interfaces compatible between components again.

The latest Weka version is version 3.9.2, released in December 2017, a minor development version. If Alice wants a stable version, she needs to choose one from several available releases, ranging from 3.6.3 to 3.6.15 (13 versions) and 3.8.0 to 3.8.2 (2 versions). The major version upgrades are 3.8.0 and 3.9.0. The former is stable and the latter is a development version. None of this information is readily available, and Alice needs to spend time analyzing the Weka documentation.

The final step in assessing choices for workflow upgrades is to decide whether to modify the software functions adopted, choosing alternatives with similar characteristics.

C. Finding similar software functions

New software functions may implement new functionalities and use new algorithms, thus opening new opportunities for the design of scientific experiments. Scientists can explore such new functions, for instance, to carry out slightly different computations in the workflow and compare the execution results. In our scenario, Alice wants to try other software functions that implement the same classification functionality but use different algorithms.

She would like to find software functions similar to the ones implemented in J48Modeler and J48Classifier. To do so, she will have to decide whether to check for alternative Weka implementations, whether to look in other software libraries. Weka functions are more likely to accept the same data format and data type for inputs and outputs, and for this reason she restricts herself to choosing from Weka options.

D. Understanding differences in software functions

To decide whether to upgrade a workflow (and which components should be updated), Alice needs to understand the differences between versions of Weka functions. Therefore, Alice needs to know what has changed in Weka since the release of version 3.6.2. Since the creation of a working workflow from scratch is a time-consuming task, especially when the workflow may contain many data preparation steps, an upgrade (and thus version comparison) is worth the effort.

Changes in functions may include function renaming, addition of input parameters, and support for different input data formats. All these kinds of changes may affect the implementation of existing workflow components.

Alice goes through the versions of the software functions in Weka 3.6.2 and 3.9.2 using the Weka command line interface, the software manual, release notes and the Javadoc documentation for help. Alice figures out that the only change needed is to update the Weka library used in the workflow. The function interfaces from Weka 3.6.2 and 3.9.2 versions are exactly the same, though this is not typically the case especially after several years have passed.

Versions may include other modifications beyond changes in software function invocation. For instance, important changes may be related to performance and accuracy, which should ideally be described in release notes.

E. Identifying known issues, bug fixes affecting software functions

A scientist may adopt a new version if it fixes some bug. However, a new version may have unintended side effects, such as affecting other functionalities in the workflow.

Alice is interested in releases containing fixes to bugs that affect her workflow. Once again, she needs to read release notes and look at the version control repository used by the Weka project. This control version repository is used to track issues and create bug fixes associated with code.

F. Creating a workflow component to explore new software functions

Alice decided that she wants to use a different classification method – the ID3 decision tree algorithm. To do this, she needs to create new workflow components to replace the ones that use the C4.5 decision tree algorithm in her workflow. For this, she needs to know how to implement a workflow component in the specific workflow system (in our case, WINGS) using the Weka software (i.e., which inputs and parameters to use, which outputs to expect, and what code to invoke). She also needs to know how to invoke the appropriate Weka function.

She chooses the ID3 Java class, which implements the desired classifier. However, she does not know which inputs to use to implement the modeling and training components.

This Java class has several possible input parameters and datasets. The combination of inputs allows it to perform different functions according to the inputs used, such as create a model using a training dataset and classify a dataset using an existing trained model. The appropriate combination of inputs which to carry out a specific function can only be found by reading the manual of the software or talking to an expert in Weka. Using Weka version 3.9.2, she can create the desired component by invoking the *ID3* java class in the package *weka.classifiers.trees*.

Then she needs to create the component´s I/O and manually map the I/O to the corresponding function I/O. This kind of

process is time-consuming and error prone. Any scientist who wants to create a new workflow component needs to go through the same process.

Alice learns about the Scikit-learn software, and wants to consider using it. This is very common, particularly in science where alternative models and libraries may provide numerous options that are often better than upgrades to older code. Alice consults the documentation of Scikit-learn and sees that it does not use the ARFF format. It is typical that using new functions from other libraries in a workflow may require additional data transformation components in the workflow. In this case, the conversion steps from CSV to ARFF in the workflow are no longer needed.

G. Requirements

In previous work we gathered several general requirements regarding workflow component metadata, workflow updates, and workflow comparisons [4]. Here, we focus on the first aspect: the requirements to capture the characteristics of the software that implements workflow components. More specifically, we focus on describing the software used in these components, and its evolution through time in order to support workflow exploration and evolution. Building on the scenarios from [4] and summarizing the scenarios above, we formulate the following requirements:

- R1 - Workflow descriptions should capture the software, software version, and functions used in the implementation of workflow components.
- R2 – Scientists should be alerted about relevant updates of software used in their workflows.
- R3 - Version descriptions should capture metadata about differences between software functions, particularly about their interfaces.
- R4 - Given a software package that can be used to create many workflow components, scientists need to easily figure out how to implement a component and how to update an existing component with newer versions of that software.
- R5 - Scientists should get a summary of changes between two given software versions to understand their differences without having to understand the history of changes associated to each version in between the old and the chosen one.
- R6 - Version descriptions should capture bug fixes and known bugs and relate them to specific software functions.

These requirements highlight the need to have metadata associated with software packages, their versions, the software functionality and functions that they implement, and the software changes done to specific functions in new versions.

The scenario we described reflects the difficulty scientists face to assess how, when and whether to upgrade their workflows or not, even when they are code-savvy. We point out that these difficulties are not specific to the choice of Weka or any other software used by scientists. Rather, the scenarios highlight that such software package repositories are designed to support code sharing and tracking, presenting technical details to programmers rather than efficiently highlighting conceptual descriptions to scientists. In other words, scientists lack a more structured and function-based representation of software to help them to design, upgrade and understand workflows. Moreover, in version control repositories, documentation is not provided in machine-readable format that can be used by a workflow system to assist the scientist in exploring and managing the evolution of a workflow.

The next section describes OntoSoft-VFF, the framework we designed to address the requirements presented in this section.

IV. OntoSoft-VFF: A Framework to Help Scientists to Explore and Update Workflows

We designed and developed OntoSoft-VFF (Ontology framework for Software Version, Function and Functionality) to address the requirements described in section III. This framework is based on a novel ontology, which is used to construct a semantic metadata catalog. OntoSoft-VFF extends OntoSoft [5], a framework composed of an ontology and a metadata catalog that aims to describe software metadata to support scientists to share and reuse software. Our extension to OntoSoft include both the novel ontology and associated services to describe software versions, functions, functionality and changes to software. OntoSoft-VFF provides the semantic information needed by scientists to explore their workflows, and to assess whether and how to update them, fully supporting the needs exemplified in the scenarios of Section III.

Figure 2 shows an overview of our ontology. We represent the elements already present in OntoSoft using the namespace (*sw*: http://ontosoft.org/software#), whereas our new ontology uses the namespace (*vff*: https://w3id.org/ontosoft-vff/ontology#). The ontology contains terms to describe:

- Software metadata: represents software, its relations with software versions, and other relations such as with operating systems, programming languages, and any software dependencies.
- Software version metadata: includes the metadata for a given software version including new functionality and functions.
- Software function metadata: includes metadata regarding functions released in software versions and their inputs and outputs.
- Software change metadata: includes metadata for representing changes in software versions over time, including known issues and bug fixes.

A major contribution of our work is to model software used in workflow components with respect to its functionality and evolution over time. In the following sections we focus on the relevant classes and relations specified in the ontology to address our requirements. Due to space limitations, we will only illustrate and describe the classes and relations related to the goals of this paper. For the same reason, we have retained here only the parts of OntoSoft that are relevant to this discussion.

The ontology is domain independent and can be extended to address specific requirements of domain scientists. For

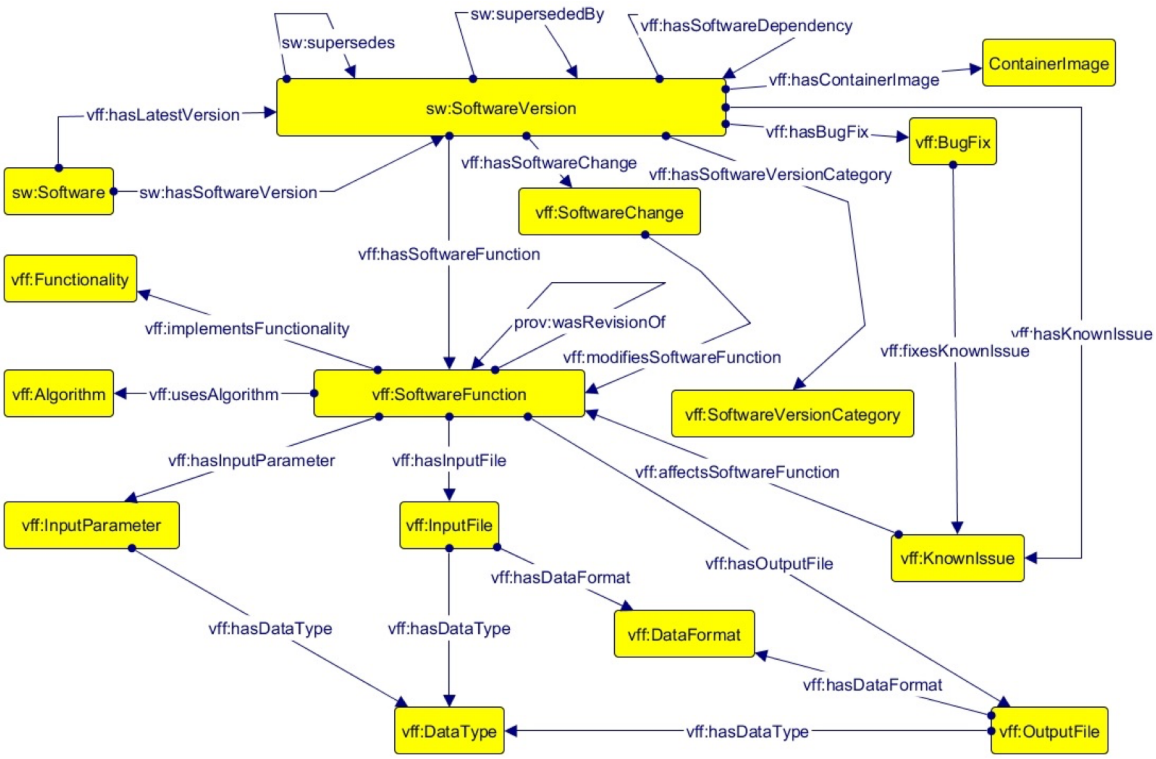

Figure 2. Diagram with the representation of the main classes and relations of our ontology.

example, we have found that for geosciences models it is important to capture environmental assumptions, variables and processes associated to modeling software [4, 9].

The ontology is available in OWL and documented in [23].

A. Describing software

Figure 3a illustrates an example of software metadata classes and properties for representing the Weka software. Weka is implemented using the Java programming language and uses a GNU license. Linux is one of the operating systems supported. Weka has the 3.9.2 version and this is its latest version.

We use the OntoSoft *sw:Software* class. It has a property *sw:hasSoftwareVersion* that relates a software with each of its versions, while *vff:hasLatestSoftwareVersion* relates a *sw:Software* to its latest version in order to provide direct access to this specific version. *sw:Software* has other properties, such as *sw:hasLicense, sw:supportsOperatingSystem* and *sw:hasImplementationLanguage*. They represent important information to know when creating a workflow component using a software.

B. Describing software versions

Figure 3b illustrates the use of the classes and properties for Weka version 3.6.2. This is a stable version of Weka released in 2010. It has the J48Classifier and ID3Classifier functions and is superseded by the 3.6.3 version.

We extended the OntoSoft *sw:SoftwareVersion* class with properties and classes to describe internal functions, versions, software dependencies, and version categories. *sw:SoftwareVersion* has properties *sw:supersededBy* and *sw:supersedes* to support navigation across software versions. To these, we added the property *vff:hasSoftwareFunction,* thereby linking *sw:SoftwareVersion* with *vff:SoftwareFunction* and thus representing functions released in a version.

We introduced the notion of category of software versions (*vff:SoftwareVersionCategory*), whose values can be: major version, minor version, stable version and development version. Categories can help scientists to decide which version to use to implement a workflow component.

vff:ContainerImage was designed to describe workflow components that use containers. Its properties such *vff:hasContainerLocation* and *vff:hasContainerInvocation* respectively specify its location in a container repository, and how to invoke the container image. This allows the isolation of a software version and its dependencies into a self-contained unit that can run anywhere independent of the environment.

C. Describing software functions

Figure 3c illustrates an example of the J48Classifier function in Weka 3.6.2 version. We recall from Section I that the difference between software and function can be subtle. A function represents a particular implementation of functionality of a software. A function is represented with the *vff:SoftwareFunction* class and implements a *vff:Functionality*, specified using the *vff:implementsFunctionality* property. A function might be implemented using a set of *vff:Algorithm*, which are specified with the *vff:usesAlgorithm* property.

In Weka, a single Java class can implement several software functions with different functionalities. Here, a function has a unique name (*vff:hasFunctionName*) and a description to help identifying its objective (*vff:hasFunctionDescription*).

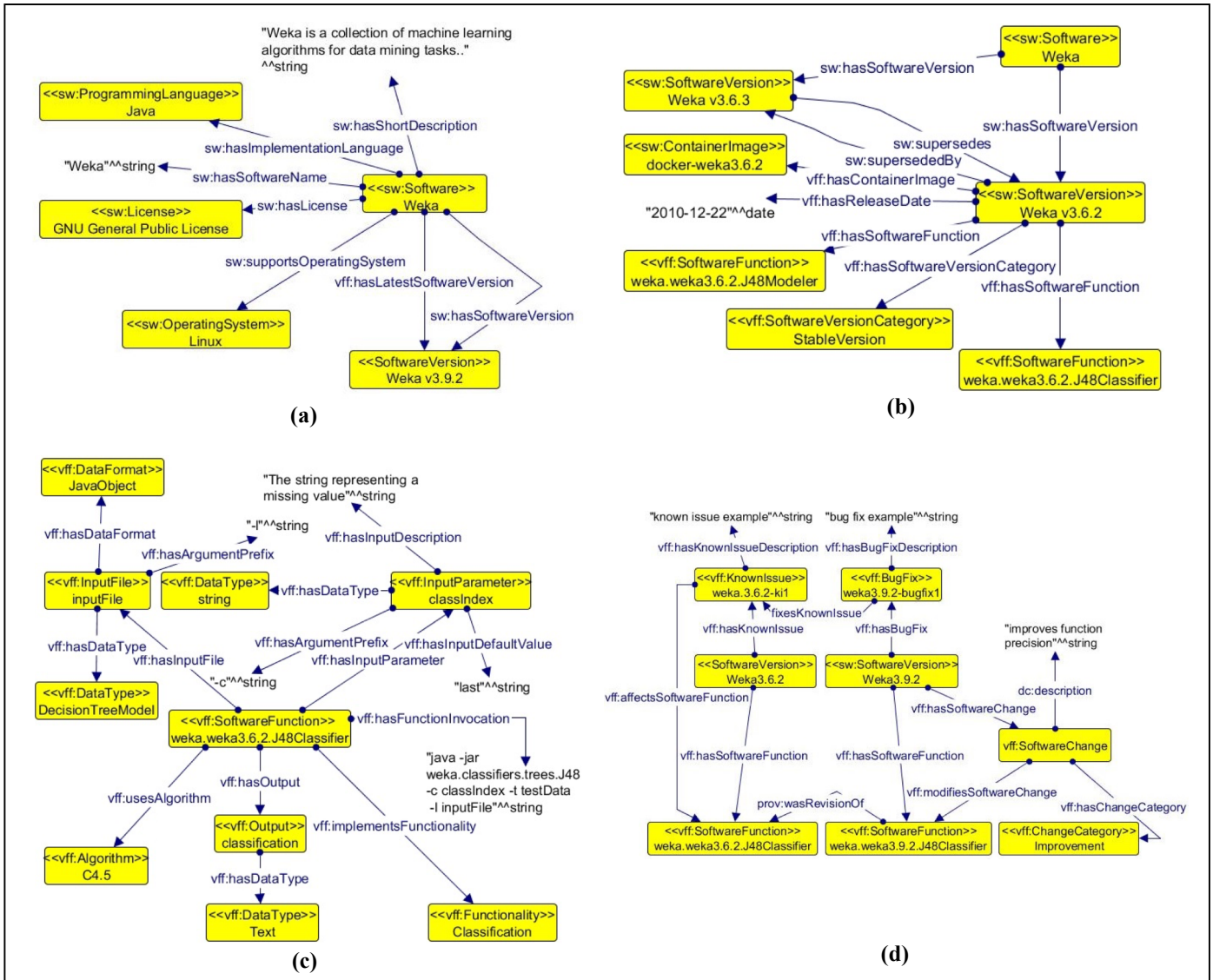

Figure 3. (a) An example of use of classes and relations from the ontology's software module to represent metadata associated with the Weka software. (b) An example of use of classes and properties from the ontology's software version module to represent metadata associated with the Weka 3.9.2 software version. (c) An example of metadata to represent the J48Classifier function from the Weka 3.6.2 version. (d) An example of metadata to represent changes to the J48Classifier function across Weka versions.

Functions have unique invocation, inputs, parameters, and outputs (*vff:hasFunctionInvocation*, *vff:InputFile, vff:InputParameter* and *vff:OutputFile)*. The inputs and outputs have a description, argument prefix and are associated with *vff:DataType* and *vff:DataFormat*. We also represent the default value associated with an input parameter (*vff:hasInputDefaultValue*), since recommended defaults are often indicated in the documentation of scientific software.

D. Describing software changes

Figure 3d shows an example of a known issue, bug fix and change associated with the *J48Classifier* function in the Weka 3.6.2 and 3.9.2 versions.

A change is defined as a modification in a software function caused by *vff:BugFix* or improvements (*vff:SoftwareChange*). Change description includes *vff:KnownIssue* as well, to represent bugs or limitations associated with *vff:SoftwareFunction* and may be fixed by *vff:BugFix* in further versions. Bug fixes and known issues have descriptions to help scientists to understand how they affect *vff:SoftwareFunction*.

V. USING ONTOSOFT-VFF TO STORE, COMPARE AND SEARCH SEMANTIC METADATA FOR SOFTWARE

This section presents the services provided by OntoSoft-VFF. We designed these services by extending the OntoSoft catalog to use the ontology extensions introduced in Section IV.

The catalog includes the following important features, taking advantage of the semantic metadata and our ontology:

- Management of software functions and evolution metadata: provides means to obtain information about software, software version, functions and changes.
- Comparison mechanism: allows the comparison between different software, software versions and functions.
- Search mechanism: allows searching for software, software version and software functions.
- Mechanism for creation of workflow components: allows the creation of components by using the metadata associated with software functions.

The source code of OntoSoft-VFF can be found in [24].

A. *Management of software functions and evolution metadata*

In OntoSoft-VFF, software developers can add metadata about software, its versions, and available functions. Developers can also provide information about know issues, relevant changes and bug fixes associated with software functions. We envision an interactive system that extracts automatically some of this information and makes the burden minimal on the developers.

When adding metadata for a new software version, our framework imports all the metadata of its previous version. The user only needs to provide information about the functions that have changed (e.g., algorithm, inputs, outputs, function name). When a function changes, OntoSoft-VFF creates a new URI for its metadata, and links it using the *prov:wasRevisionOf* property from the W3C PROV standard [22]. Through this URI, a workflow component can refer to the specific version of a function used in its implementation.

By adding bug fixes in new versions released, the user can provide information about known issues and associate them with specific functions in previous versions.

B. *Comparison across versions and functions*

The OntoSoft catalog allows the comparison of software via its metadata. We extended this to allow the comparison of software versions and functions as well. Our extension provides a simple comparison for software versions based on the functions they implement and the software version categories.

Function comparison is done by using metadata about functionality, algorithms, data types and data formats for inputs and outputs of functions, as well as relevant changes to functions such as bug fixes or improvements or known issues. Functions can be compared to other functions in the same software version, or to functions belonging to different software versions. This helps scientists understand the changes and differences in functions over time.

Figure 4 shows the comparison of functions using metadata of the functions ID3Classifier and J48Classifier in Weka version 3.6.2. Due to space limitations, we only show functionality, algorithm, invocation line, and input files. As we can see, these functions have the same inputs and functionality. However, they use different algorithms and distinct function invocations, since they are implemented by different Java classes in Weka.

weka.weka3.6.2.ID3Classifier	weka.weka3.6.2.J48Classifier
[OPTIONAL] Functionality	
Classification	Classification
[OPTIONAL] What is the algorithm used in this function?	
ID3 Decision Tree	C4.5
[OPTIONAL] What is the invocation line for this function?	
java weka.classifiers.trees.Id3 -T testData -l inputFile -c classIndex > classification	java weka.classifiers.trees.J48 -T testData -l inputFile -c classIndex > classification
[OPTIONAL] What input files does this function require?	
Input File Name: inputFile **Input File Description:** Sets input model file. **Input File Argument:** -l **Input File Data Type:** Decision Tree Model **Input File Data Format:** Java Object	**Input File Name:** inputFile **Input File Description:** Sets model input file. **Input File Argument:** -l **Input File Data Type:** Decision Tree Model **Input File Data Format:** Java Object

Figure 4. Example of comparison between the ID3Classifier and J48Classifier functions.

C. *Search*

The search capability allows scientists to find software, software versions, and software functions using their metadata. Search parameters include keywords related to the software and function names, functionality, license, data formats and data types of function inputs and outputs.

D. *Creation of workflow components*

OntoSoft-VVF facilitates the creation of workflow components using the semantic metadata associated with a software function. This helps scientists to create a workflow component from scratch. This mechanism is implemented as an external tool with access to the software function metadata. Our catalog exports the metadata in JSON format and also allows the use of a SPARQL endpoint to query the software metadata for the creation of workflow components. Other work can take advantage of OntoSoft-VFF to implement a similar mechanism for a different workflow management system.

We demonstrate this mechanism by integrating our catalog with the WINGS workflow system. Because WINGS uses semantic workflows, this system can take full advantage of semantic metadata available in OntoSoft-VFF´s catalog to create workflow components. Our framework automatically creates in WINGS the inputs and outputs for the new workflow component and maps them to the software function's inputs and outputs. Using the metadata available, OntoSoft-VFF automatically configures the interface of the new component, including data types, and creates the invocation code to the function in the workflow component. We assume that there is a container image available from an online repository for each software version, which can be used to invoke the software function, thus avoiding the burden to install and configure the software dependencies. This tool creates the ID3Classifier component using the function with the same name from the Weka 3.9.2 version. This component can be used to replace the J48Classifier component in a workflow as in our scenarios.

VI. VALIDATION

To validate the ontology and services of OntoSoft-VFF, we demonstrate the ability of our framework to answer a series of competency questions on software functions and evolution, and the use of software functions in workflow components. The competency questions have been drawn from the requirements outlined in Section III.

We designed queries to be evaluated against the structured metadata captured for the scenarios presented in Section III. We present a description of each query, their translation into SPARQL, and the results obtained by evaluating them. The competency questions guided the development of our ontology and justified the creation of classes and properties. We use the namespace (*ex*: https://w3id.org/ontosoft-vff/example) to refer to the instance of our ontology in SPARQL.

***Query 1**: Given a software function invoked by a workflow component implementation, what is the software and software version of this function?*

This query is useful to identify the software and software version of a function used to implement a workflow component. Retrieving metadata about software used in a workflow implementation addresses requirement R1.

Here, we are interested in retrieving the software function (*ex:weka3.6.2-J48Classifier*) responsible for implementing the *J48Classifier* workflow component. Specifically, the results point out that this software function is from the Weka software (*ex:Weka*) and was released in the Weka version 3.6.2 (*ex:Weka3.6.2*).

The SPARQL used for answering this query can be formulated as follows:

```
select ?sw ?swVersion where {
 ?swVersion rdf:type sw:SoftwareVersion ;
    vff:hasSoftwareFunction ex:weka3.6.2-J48Classifier .
 ?sw rdf:type sw:Software ;
   sw:hasSoftwareVersion  ?swVersion . }
```

***Query 2**: Are there any newer versions for a given function?*

This query is useful to identify new versions of a given function used in a workflow component. Some software versions may not change a software function; thus, it is not the case of finding new software versions. This query retrieves a new function version of a given software function, which can be further compared to understand their differences. Retrieving metadata about software version releases addresses requirement R2.

Here, we are interested in showing the newest version of the J48Classifier function (*ex:weka3.6.2-J48Classifier*). The query results point out that the J48Classifier function has a new version (*ex:weka3.9.2-J48Classifier*) in Weka 3.9.2 version (*ex:weka3.9.2*).

The SPARQL query used for answering this query can be formulated as follows:

```
select ?swVersionNew ?swFunctionNew where {
  ?swVersionNew rdf:type sw:SoftwareVersion ;
     vff:hasSoftwareFunction ?swFunctionNew .
  ?swFunctionNew rdf:type sw:SoftwareFunction ;
     prov:wasRevisionOf  ex:weka3.6.2-J48Classifier . }
```

***Query 3**: What are the differences between two versions of a given software function?*

This query is useful to detect the differences between two version of a software function, particularly their interfaces, to use that information to upgrade a workflow component. Detecting differences between two versions of a software function addresses requirement R3.

Here, we are interested in the J48Classifier function from the 3.6.2 version (*ex:weka3.6.2-J48Classifier*) and the 3.9.2 version (*ex:weka3.9.2-J48Classifier*). We run a separate query for each software function and then compare their results to compare the functions I/O. The results point out that there is no difference between their interfaces (i.e., their inputs and outputs).

The SPARQL used for answering this query can be formulated as follows:

```
select ?inputName ?inputDataFormat ?inputDataType
?inputParamName ?inputParamType ?outputName
?outputDataFormat ?outputDataType where {
ex:weka3.6.2-J48Classifier rdf:type vff:SoftwareFunction ;
    vff:hasInputFile  ?inputFile ;
    vff:hasInputParameter ?inputParam ; vff:hasOutputFile ?output .
 ?inputFile vff:hasInputDataFormat ?inputDataFormat ;
   vff:hasInputDataType ?inputDataType ;
   vff:hasInputName ?inputName .
 ?inputParam vff:hasInputParameterDataType ?paramType ;
     vff:hasInputParamName ? inputParamName .
 ?output vff:hasOutputDataFormat ?outputDataFormat ;
     vff:hasOutputDataType  ?outputDataType ;
     vff:hasOutputName ?outputName . }
```

***Query 4**: Are there any similar functions to a given function in newer software versions?*

This query is useful to find similar functions in newer software versions based on their functionalities. We designed the query to find software functions that implement the same functionality or use the same algorithm than a given software function used in a workflow component. We can filter the functions by software and software version or find software function across different software. Detecting differences between two software versions, particularly about new software functions available addresses requirement R4.

Here, we are interested in finding a similar function to J48Classifier (*ex:weka3.6.2-J48Classifier*) that implements the same functionality in the same software version (*ex:weka3.6.2*). The query results point out that the *ID3Classifier* function (*ex:weka3.6.2-ID3Classifier*) implements the same functionality the *J48Classifier* function does.

The SPARQL used for answering this query can be formulated as follows:

```
select ?swFunction where {
  ex:weka3.6.2-J48Classifier rdf:type vff:SoftwareFunction ;
     vff:implementsFunctionality ?functionality .
  ?swFunction rdf:type vff:SoftwareFunction ;
     vff:implementsFunctionality ?functionality .
  ex:weka3.9.2 vff:hasSoftwareFunction ?swFunction .
  FILTER(ex:weka3.9.2-J48Classifier != ?swFunction) . }
```

Query 5: *How to invoke a given software function?*

This query is useful to implement the invocation code of a workflow component based on the specification of an existing software function. Retrieving metadata about software functions, particularly their invocation code addresses requirement R5.

Here, we are interested in retrieving metadata associated with invocation of the ID3Classifier function (*ex:weka3.6.2-ID3Classifier*), such as function invocation and container invocation. By retrieving this information, we can create the invocation code in a workflow component. The query results point out that the function invocation is "*java -jar weka.classifiers.trees.Id3 -T testData -l inputFile -c classIndex > classification"* and the container invocation is *"docker run lucasaugustomcc/weka3.6.2"*.

The SPARQL used for answering this query can be formulated as follows:

```
select ?functionInvocation ?containerInv  where {
  ex:weka3.9.2-ID3Classifier rdf:type vff:SoftwareFunction ;
     vff:hasSoftwareFunctionInvocation  ?functionInvocation .
  ?swVersion rdf:type sw:SoftwareVersion ;
      vff:hasContainerImage ?containerImg ;
      vff:hasSoftwareFunction ex:weka3.9.2-ID3Classifier .
  ?containerImg vff:hasContainerImageInvocation  ?containerInv .  }
```

Query 6: *a) Are there any known issues that affect a given software function?*

This query is useful to find out known issues that can affect the performance or results of software functions.

Here, we are interested in retrieving known issues associated with the ID3Classifier function (*ex:weka3.6.2-ID3Classifier*). The query results point out that no known issues are associated with this function.

The SPARQL used for answering this query can be formulated as follows:

```
select ?bug ?bugDescription where {
  ?bug rdf:type sw:KnownIssue ;
      vff:hasKnownIssueDescription ?bugDescription ;
      vff:affectsSoftwareFunction ex:weka3.6.2-J48Classifier . }
```

b) Are there any important changes associated with new versions of a given software function?

This query is useful to find out which software version they should upgrade a workflow component to take advantage of improvements associated with versions of software functions.

Here, we are interested in retrieving changes associated with the J48Classifier function. The query results point out that there are no bug fixes associated with the J48Classifier function in Weka 3.9.2.

The SPARQL used for answering this query can be formulated as follows:

```
select ?bugFix ?bugFixDescription where {
  ?bugfix rdf:type vff:BugFix ;
      vff:hasBugFixDescription ?description ;
      vff:fixesKnownIssue ?knownIssue .
  ?knowIssue
      vff:affectsSoftwareFunction ex:weka3.6.2-J48Classifier . }
```

Retrieving metadata about known issues and bug fixes associated with different versions of software functions used in a workflow component addresses requirement R6.

In summary, these queries show that the requirements are fully supported by OntoSoft-VFF, and that when they are not supported directly the framework provides the information necessary to address them. OntoSoft-VFF can answer questions that have been traditionally answered by scientists with great effort. The competency questions and the results obtained by evaluating the queries can be found in [25].

VII. Conclusions and Future Work

This paper presented OntoSoft-VFF, a semantic software catalog designed and developed to help scientists to manage workflow exploration and evolution, while they update or investigate alternatives for their computational experiments. OntoSoft-VFF relies on an ontology we designed to capture software versions, functionality, and functions and their evolution over time. This ontology is used in the construction of OntoSoft-VFF's underlying semantic metadata for software.

We showed that when a workflow is semantically linked to such metadata, scientists can explore the workflow to understand its evolution, and to compare among several software implementations to select one to implement a workflow's component. While related work is mostly concerned with workflow design, evolution, or provenance information, our goal is to help scientists to understand the evolution of the software used in the workflow components.

OntoSoft-VFF was built to meet requirements found through exploration of scenarios based on our experience using a variety of machine learning software libraries as well as diverse hydrology models. We demonstrate through competency questions that OntoSoft-VFF successfully meets those requirements. The competency questions and the scenarios are additional contributions of our work, since they describe very common scientific practices which are taken for granted and thus seldom explicitly formulated.

There are several possibilities for extending our work. One of them is to further explore the scenarios and competency questions in order to set up a benchmark for research on workflow evolution. A limitation of OntoSoft-VFF is that the addition of software metadata was manually done. This could be done semi-automatically in the future. Also, we plan to integrate OntoSoft-VFF with a workflow system to support scientists to efficiently update their workflows as the underlying application software evolves, and to easily explore new designs for their computational experiments. Finally, we plan to align to other ontologies such as the SWO, which contains thousands of instances.

ACKNOWLEDGMENTS

This work was supported in part by the Sao Paulo Research Foundation (FAPESP) under grants 2017/03570-3, 2014/23861-4 and 2013/08293-7, in part by a grant from the Defense Advanced Research Projects Agency under award W911NF-18-1-0027, and, in part by a grant from the US National Science Foundation under award ICER-1440323 and ICER- 1632211 (EarthCube RCN IS-GEO).

REFERENCES

[1] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman, "Wings: Intelligent workflow-based design of computational experiments," IEEE Intelligent Systems, vol. 26, no. 1, pp. 62–72, 2011.

[2] J. Zhao, J. M. Gomez-Perez, K. Belhajjame, G. Klyne, E. Garcia-Cuesta, A. Garrido, K. Hettne, M. Roos, D. De Roure, and C. Goble, "Why workflows break—understanding and combating decay in taverna workflows," in E-Science (e-Science), 2012 IEEE 8th International Conference on. IEEE, 2012, pp. 1–9.

[3] D. Koop, C. E. Scheidegger, J. Freire, and C. T. Silva, "The provenance of workflow upgrades," in International Provenance and Annotation Workshop. Springer, 2010, pp. 2–16.

[4] L. A. M. C. Carvalho, B. T. Essawy, D. Garijo, C. B. Medeiros, and Y. Gil, "Requirements for supporting the iterative exploration of scientific workflow variants," in Proceedings of the Workshop on Capturing Scientific Knowledge (SciKnow), held in conjunction with the ACM International Conference on Knowledge Capture (K-CAP), Austin, Texas, 2017.

[5] Y. Gil, V. Ratnakar, and D. Garijo, "Ontosoft: Capturing scientific software metadata," in Proceedings of the 8th ACM International Conference on Knowledge Capture. Palisades, NY: ACM, 2015.

[6] Y. Gil, P. A. Gonzalez-Calero, J. Kim, J. Moody, and V. Ratnakar, "A semantic framework for automatic generation of computational workflows using distributed data and component catalogues," Journal of Experimental & Theoretical Artificial Intelligence, vol. 23, no. 4, pp. 389–467, 2011.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The Weka data mining software: an update," ACM SIGKDD explorations newsletter, vol. 11, no. 1, pp. 10–18, 2009.

[8] J. Malone, A. Brown, A. L. Lister, J. Ison, D. Hull, H. Parkinson, and R. Stevens, "The software ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation," Journal of biomedical semantics, vol. 5, no. 1, p. 25, 2014.

[9] D. Garijo, D. Khider, Y. Gil, L. A. M. C. Carvalho, B. T. Essawy, S. Pierce, D. H. Lewis, V. Ratnakar, S. Peckham, C. Duffy, and J. Goodall, "A semantic model catalog to support comparison and reuse," Proceedings of the 9th International Congress on Environmental Modelling and Software (iEMSs), Fort Collins, Colorado, USA, 2018.

[10] D. Oberle, S. Grimm, and S. Staab, "An ontology for software," in Handbook on ontologies. Springer, 2009, pp. 383–402.

[11] R. Conradi and B. Westfechtel, "Version models for software configuration management," ACM Computing Surveys (CSUR), vol. 30, no. 2, pp. 232–282, 1998.

[12] E. C. Withana, B. Plale, R. Barga, and N. Araujo. "Versioning for workflow evolution." In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 756-765. ACM, 2010.

[13] D. Garijo, S. Kinnings, L. Xie, L. Xie, Y. Zhang, P.E. Bourne and Y. Gil. Quantifying reproducibility in computational biology: the case of the tuberculosis drugome. PloS ONE, 8(11), 2013.

[14] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A. and Li, P. 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, *20*(17), pp.3045-3054.

[15] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B. and Mock, S., 2004, June. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on* (pp. 423-424). IEEE, 2004.

[16] Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J. and Silva, C.T., 2008, June. Querying and re-using workflows with VsTrails. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1251-1254). ACM, 2008.

[17] Pedregosa, F., Varoquaux G., Gramfort, A. Michel V., Thirion, B. Grisel O., Blondel, M. Prettenhofer, P. Weiss, R. Dubourg, V. Vanderplas, J. Passos, A. Cournapeau, D. Perrot, M., and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 12: 2825–2830

[18] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[19] Hauder, M., Gil, Y., and Y. Liu. A Framework for Efficient Text Analytics through Automatic Configuration and Customization of Scientific Workflows. Proceedings of the Seventh IEEE International Conference on e-Science, Stockholm, Sweden, 2011.

[20] Sethi, R. J., Jo, H., and Y. Gil. Reusing Workflow Fragments Across Multiple Data Domains. Proceedings of the Seventh Workshop on Workflows in Support of Large-Scale Science (WORKS'12), held in conjunction with SC 2012, Salt Lake City, Utah, 2012.

[21] Gil, Y.; Yao, K.; Ratnakar, V.; Garijo, D.; Steeg, G. V.; Szekely, P.; Brekelmans, R.; Kejriwal, M.; Luo, F.; and Huang, I. P4ML: A Phased Performance-Based Pipeline Planner for Automated Machine Learning. Proceedings of Machine Learning Research, ICML 2018 AutoML Workshop, 2018.

[22] Moreau, L.; Missier, P.; Belhajjame, K.; B'Far, R.; Cheney, J.; Coppens, S.; Cresswell, S.; Gil, Y.; Groth, P.; Klyne, G.; Lebo, T.; McCusker, J.; Miles, S.; Myers, J.; Sahoo, S.; and Tilmes, C. 2013. PROV-DM: The PROV Data Model. World Wide Web Consortium (W3C) Recommendation, 2013.

[23] L. A. M. C. Carvalho, D. Garijo, C. B. Medeiros, and Y. Gil, The OntoSoft-VFF Ontology. Version 1.0.0. Available from: https://w3id.org/ontosoft-vff/ontology.

[24] L. A. M. C. Carvalho, D. Garijo, C. B. Medeiros, and Y. Gil, The OntoSoft-VFF Source Code. Version 1.0.0. Zenodo. http://doi.org/10.5281/zenodo.1414544.

[25] L. A. M. C. Carvalho, D. Garijo, C. B. Medeiros, and Y. Gil, The OntoSoft-VFF Evaluation: Competency Questions. Version 1.0.0. Zenodo. http://doi.org/10.5281/zenodo.1414552.