IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 20, 2018, accepted October 14, 2018, date of publication October 25, 2018,
date of current version November 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2878043

Behavioral Analysis of Scientific Workflows With Semantic Information

JAVIER FABRA[1], MARÍA JOSÉ IBÁÑEZ[1], PEDRO ÁLVAREZ[1], AND JOAQUÍN EZPELETA[1]
Aragón Institute of Engineering Research, Department of Computer Science and Systems Engineering, Universidad de Zaragoza, 50009 Zaragoza, Spain
Corresponding author: Javier Fabra (jfabra@unizar.es)

This work was supported in part by the Spanish Ministry of Economy, Industry and Competitiveness under Project TIN2017-84796-C2-2-R and in part by the Aragonese Department of Innovation, Research and Universities under Project DisCo-T21-17R.

ABSTRACT The recent development in scientific computing related areas has shown an increasing interest in scientific workflows because of their abilities to solve complex challenges. Problems and challenges that were too heavy or time consuming can be solved now in a more efficient manner. Scientific workflows have been progressively improved by means of the introduction of new paradigms and technologies, being the semantic area one of the most promising ones. This paper focuses on the addition of semantic Web techniques to the scientific workflow area, which facilitates the integration of network-based solutions. On the other hand, a model checking technique to study the workflow behavior prior to its execution is also described. Using the unary resource description framework annotated Petri net formalism, scientific workflows can be improved by adding semantic annotations related to the task descriptions and workflow evolution. This technique can be applied using a complete environment for the model checking of this kind of workflows that is also shown in this paper. Finally, the proposed methodology is exemplified by its application to a couple of known scientific workflows: the first provenance challenge and the InterScan protein analysis workflow.

INDEX TERMS Petri nets, prediction analysis, RDF, scientific workflows, semantics.

I. INTRODUCTION

In the last years, scientific computing workflows have gained a lot of interest in different areas related to science and human life. Scientific workflows are a special type of workflows which often underlies many large-scale complex e-science applications such as climate modeling, structural biology and chemistry, medical surgery or disaster recovery simulation, among others. Scientific workflows have been progressively improved by means of the introduction of new paradigms and technologies in order to achieve more complex challenges. The focus of this work is on the incorporation of semantic Web techniques to the scientific workflow area. This has improved and made more flexible the description and discovery of services, resources or workflows ([1]–[5]), the composition of services and resources ([3], [6], [7]), or the analysis of provenance data ([8]–[11]). The addition of semantic aspects allows scientists to more efficiently and flexibly browse, query, integrate and compose relevant cross discipline datasets and services [1].

Scientific workflow execution is expensive in terms of resource usage as well as a time consuming activity. For this reason, it is of special interest to be provided with tools and techniques making possible the analysis of the workflow behavior prior to its execution [12], [13]. The aim of such analysis would be to ensure a correct behavior as well as facilitating having a very efficient resource utilization from both budget and time points of view. In the end, it would be a waste of time and money to realize after the execution of a long task that the output has not the correct information to feed the next task. The result of the analysis should allow predicting the quality of the results and also identifying those parameters suitable to get the expected outcome. The introduction of semantic aspects in workflows increases flexibility in the sense that it allows considering third party task implementation as a natural approach. The drawback is that such inclusion requires new models and analysis techniques, able to deal with such semantic aspects, to be considered. This is the aim of this paper.

Regarding modeling techniques in scientific workflows, one of the first research proposals that take advantage of semantic Web technologies in the scientific workflows area is [1]. This approach combines metadata support with (Web) services within a framework that supports scientific workflows and shows how semantics descriptors are incorporated to the services technology. In the same line [2], [14] concentrate on the advantages of the use of semantic aspects

in scientific workflows. Their conclusions are that providing information of the context using semantic annotations can be useful for the discovery and execution of workflows, for the reuse of workflows. Reference [3] focuses on the inclusion of quality information in science workflows. It proposes an approach based on RDF annotations of OWL DL ontologies. [4] presents an approach that supports reasoning in scientific workflows and uses this approach for resource discovery. The project myExperiment [5] is a social network based on semantic Web technologies that allows finding, using and sharing scientific workflows and other research objects. myExperiment has many different types of workflows, such as Taverna, Galaxy, Rapid Miner, Bio Extract, and Kepler. As a part of this project, [15] proposes an OWL DL ontology allowing its data to be published in a standard RDF format allowing an expressive discovery.

As in any system, an important aspect to be considered when preparing scientific workflows for execution is *correctness*, thinking of it as ensuring that some important properties will be satisfied. The inherent distributed nature of workflow executions, together with the required time and computing resources, makes ensuring a good behavior a necessity. Checking proper execution is a complicated task, which becomes even more difficult when task are going to be executed by third party services. In the case of workflow systems, model checking has been one of the most important technologies, mainly when control-flow aspects have to be considered. LTL model checking is used in [16] for the case of workflows in the e-commerce domain. The analysis is carried out from a structure directly extracted from the Java code implementing the services. On the other hand, [17] carries out CTL model checking for a class of Petri nets with specific building blocks for the workflow domain. A different level of complexity is added when data and data flow must be considered. And this is the case we are considering: workflow tasks will correspond to service invocations, which we must assure are adequate according to their published specifications. Also in this aspect model checking can be considered. Reference [18] presents a review on how model-checking technologies can be used for the verification and validation of web services. In [19] BPEL processes are formalized in terms of interacting agents. The model is translated into Promela and LTL based model-checking is applied for the verification of behavioral properties. A similar approach can be found in [20], but the authors concentrate on providing models for a big set of workflow control patterns, so that business processes can also be checked for behavioral properties. A very interesting solution is the one proposed in [12], which presents an framework based on the modeling of control-flow aspects by means of asynchronous π–calculus, using λ–calculus for data flow aspects. The system is then analyzed using a CTL model checker. The framework is based on separating the analysis of control and data properties. Reference [21] uses LTL model checking for the formal verification and validation of existing web applications, generating automata models from execution traces of a web application. Another very interesting approach is presented in a recent work [13], where authors explore the soundness of Workflow nets with Data Constraints (WFDC-nets) to describe different correctness requirements. To do that, the reachability graphs of WFDC nets are processed with a set of algorithms related to different properties that the authors propose. The unfolding technique of Petri nets applied to the soundness property is also applied to Workflow nets in [22]. In that work, authors propose an algorithm to generate a finite prefix of the unfolding of a Workflow net that overcomes the state explosion problem and allows to verify the soundness property as well. To this end, authors propose the use of the branching-process technique as an effective model-checking technique.

In this paper we are going to show how the class of models and analysis techniques (the Unary RDF Annotated Petri Net formalism, namely U-RDF-PNs, and the RDF Model Checking Analysis Technique, respectively) in [23] can be applied to the domain of scientific workflows, allowing the scientists community to take advantages of the new semantic technologies and facilitating sharing workflows and tasks as well as reasoning about the results and behaviors. The starting point of the analysis is a scientific workflow modeled by means of a Petri net in which transitions correspond to the description of service invocations specified by means of semantic information (RDF graphs) describing how the input and output parameters are, as well as some pre and post-conditions related to the service input and output. The aim is to prove whether some behavioral properties are verified or not (such as the possibility of the workflow to terminate or reaching some desired state, for instance) from the model and an initial marking corresponding to the initial data provided (described by means of RDF graphs). Petri net structure provides with the required means to deal with control-flow aspects, while (RDF based) semantic technologies provide with the required flexibility for service specification and data integration. The usual semantics of Petri net models provides with the means for a very natural integration of data and control flow aspects. Once the model is built, behavioral analysis can be done using a CTL-based model checker specifically developed for the U-RDF-PN formalism.

The remainder of this paper is organized as follows. Section II introduces the main ideas in the paper by means of a first example based in the First Provenance Challenge. The way to use semantic Web techniques in order to describe the data associated with each activity in a scientific workflow is also depicted in that section. The Resource Description Framework, RDF, is then introduced in Section III. Knowing about the main concepts of RDF is required to understand the nature and foundations of the proposed approach. After that, Section IV details how to use the Parametric Unary RDF Annotated Petri Net formalism and the Parametric RDF Model Checking Technique in order to model and analyze scientific workflows. The way the behavioral analysis can be used is then introduced in Section V. The implementation of a framework for model checking supporting the presented

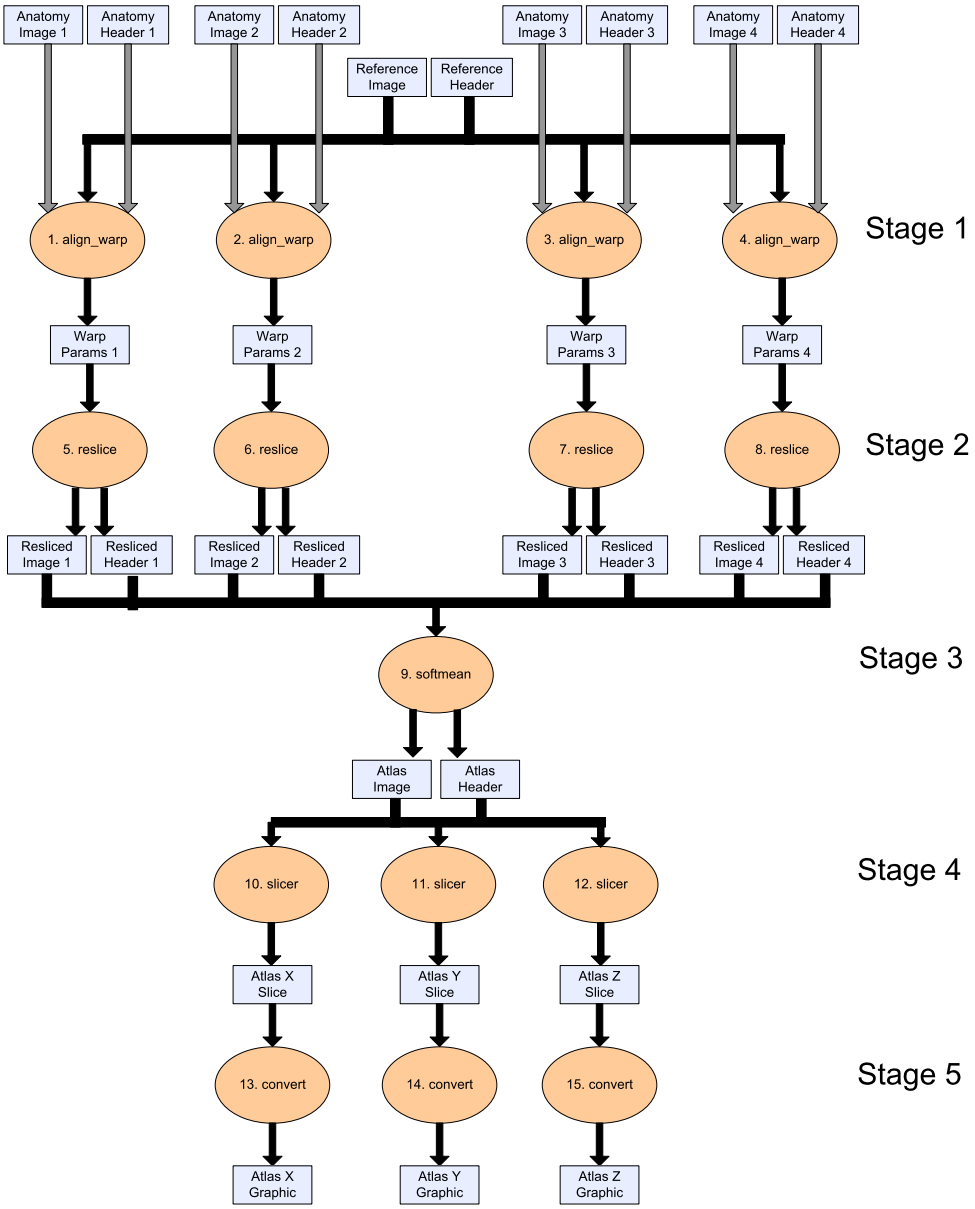

FIGURE 1. Workflow of the *First Provenance Challenge*.

approach is also provided there. Another application scenario related to the InterProScan analysis of a protein sequence is depicted in Section VI. Finally, Section VII concludes the paper and addresses some limitations of the approach as well as future research directions.

II. INTRODUCING SEMANTICS IN SCIENTIFIC WORKFLOWS

Let us introduce the main concepts of the proposed approach by means of an introductory example. The First Provenance Challenge [24] is an experiment from the area of Functional Magnetic Resonance Imaging (fMRI). Its aim is to create *population-based brain atlases* from the Functional Magnetic Resonance Imaging Data Center's archive. Currently, this archive is hosted in a cloud-based environment in the neuroimaging data repository at Neuroimaging Informatics Tools and Resources Clearinghouse[1] of high resolution anatomical data.

Figure 1 depicts the workflow of this example. As shown, the workflow specifies the set of processing steps to be carried out over the input in order to obtain the final result. In each stage there is a specific procedure which process the input data and generates the output, which is then used as an input for the next stage. Let us detail these parameters and steps. The input of the workflow are a set of four brain images (`Anatomy Image` 1 to 4) and a single reference

[1] http://www.nitrc.org

brain image (`Reference Image`), which is used with each previous brain image as input for the different stages of the workflow. For each image, besides the pixel image itself there is the metadata information (`Anatomy header` 1 to 4 and `Reference Header`).

Let us now briefly depict the different stages of the workflow. The first stage requires as input parameters a brain image, the reference image and also the meta-data information attached to the brain image. At this stage, the `align_warp` process compares the reference image to determine how the new image should be warped, this is, the position and shape of the image adjusted, to match the reference brain. As an output, this procedure generates the optimal warp parameter set which defines the spatial transformation to be performed in the next stage (`Warp Params` 1 to 4).

At second stage, the transformation of the image is performed by the `reslice` process using each warp parameter set. This creates a new version of the original brain image with the configuration defined in the warp parameter set. The output is a re-sliced image.

It is important to remark that these first two stages can be executed in parallel. Once all the execution flows (one per each input image) have finished, all the re-sliced images are averaged into one single image using the `softmean` procedure at the third stage. As a result, an atlas image and its attached meta-data information are generated.

Then, the averaged image is `sliced` for each dimension (x, y and z) to give a 2D atlas along a plane in that dimension, taken through the center of the 3D image. The output is an atlas data set, which is then converted into a graphical atlas image using the ImageMagick utility `convert` (fourth and fifth stages, respectively).

The specification of a scientific workflow must consider two important aspects. On the one hand, which are the specific operations to be executed. On the other one, the set of possible orderings in which the operations have to be made. In the general case of scientific workflows the ordering constrains are mainly due to the data dependencies. In this example, the combination of the images must be done once the `align_warp` and `reslice` operations have finished for every input image, for example.

With respect to the specification of each operation, different situations should be considered. In this specific example we are dealing with a very closed workflow domain, where the operations are taken from known and detailed libraries, so that it is quite easy to invoke a procedure: the scientist knows how the procedures must be invoked, as well as how the results of each invocation are. Therefore, it is easy to ensure that the input parameters will give a set of correct results. The First Provenance Challenge is a good example which reflects the traditional elements in scientific computing scenarios, and which will properly introduce our approach.

However, it is a current (and desirable) trend to look for processing tools and services over the Internet, integrating them into a workflow, rather than locally develop and execute

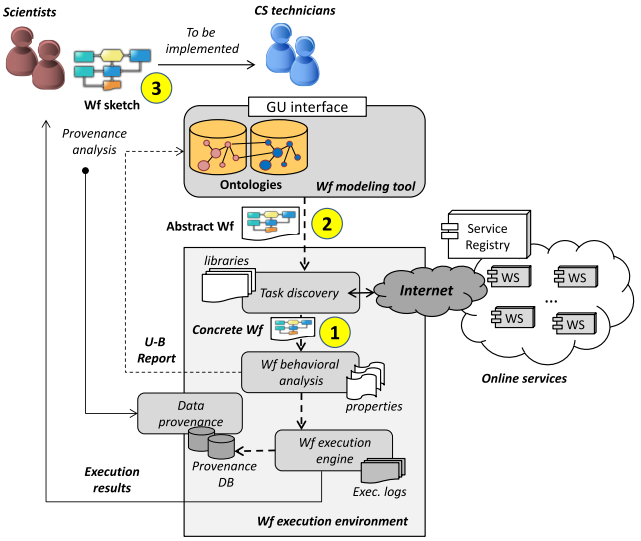

FIGURE 2. Modeling and execution of service-oriented workflows enhanced with semantic descriptions.

every required service. In this sense, service oriented computing allows the development of new added value services by integrating net-based services. The integration of both Web services and semantic Web technologies provide with the required help: Web services facilitate the means to access services through the Internet, while semantics provides with the means to standardize the description of the service itself and their properties so that automatic tools can test whether it is the appropriate one for the task to be accomplished (see [14] for some example in the biomedical problem domain).

Figure 2 shows the process and tools needed to design, program and execute service-oriented workflows enhanced with semantic information. Initially, scientists represent their experiment by means of a *sketch* that determines the flow of operations and data involved in them. Then, this non-technical representation is translated into a workflow-based model by computer science technicians. Workflow modeling tools provide the functionality to help technicians in the design of these workflows. *Domain ontologies* will be used to describe the inputs and outputs of the workfow, the requirements and data involved in each processing step, and the control and data dependencies of the workflow. The result is an *abstract workflow* that cannot be directly executed yet.

Service-oriented computing allows the development of new *workflow execution environment* able to integrate net-based services. Since there can be many different services, abstract workflows can be instantiated by means of an automatic process in which viable services are located and selected from some service registries [25]. This automatic *service discovery and integration* process is compatible with the use and invocation of libraries and procedures provided by scientists. Semantic descriptions provide with the means to automate these tasks and compile a *concrete workflow*. Before executing the resulting workflow, it must be analyzed in order to detect unexpected and erroneous behaviors.

This analysis consists of verifying a set of behavioral properties which will guarantee the correct execution of the programmed workflow. Finally, this workflow is executed by an engine which will return the corresponding results to scientists and store the provenance information of each execution into a internal data base. This information will help programmers to track workflow data throught all transformations and scientists to reproduce their experiments and analyze their defects.

Nowadays, the starting point of our proposal is a concrete workflow enhanced with semantic information (*label with number 1* in Figure 2). In this paper we focus on two open challenges: i) to know if a *discovered* and provided operation (service) fits our needs for an specific task to be done for some data; and ii) to be able to ensure that an entire workflow formed using (or composed of) a set of such external operations could terminate properly. To help in dealing with these two aspects, the presented approach relies on two different technologies: i) semantics in order to have a rich and flexible way of describing not only the processing capabilities of a task service, but also the required inputs and outputs; and ii) a class of formal models, called *Unary RDF Petri nets* (U-RDF-PN) for the modeling and analysis of workflows using RFD-based semantic information, which will be able to answer questions about the workflow's behavior.

Nevertheless, ideally new modeling tools should be developed or adapted to make the programming of these semantic-based workflows easier (*label with number 2* in Figure 2), and new model-driven methodologies should be defined to automate the translation of high-level representations to workflow models (*label with number 3* in Figure 2).

Let us now introduce the main semantic concepts and how semantic Web techniques can be used to describe input, outputs, pre- and post-conditions in the tasks that conform the workflow specification.

III. THE RESOURCE DESCRIPTION FRAMEWORK

The Semantic Web is based on the idea of adding semantic and ontological metadata to the World Wide Web components. This additional information, which describes the content, the meaning and the relations among data, must be provided in a formal way, so that they can be automatically processed by machines.

The *Resource Description Framework* (RDF for short) is a language designed to represent information about resources on the Web (the author of a Web page or some license information, for instance) [26]. However, by generalizing the concept of a *Web resource*, RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved from the Web. Examples of this usage include information about items available from on-line shopping facilities (for example, information about specifications, prices, and availability) or the description of a Web user's preferences for information delivery [26].

In the approach presented in this paper, RDF is used to describe information about the inputs and outputs of each task represented in the workflow, as well as information about the data flows which are carried out through the workflow. The RDF model requires the description of resources in terms of properties and values. This is done by means of the creation of simple statements about resources which consist of three elements: subject, property, and object. For example, for the statement *An image has a matrix called Matrix1*, the subject is *An image*, the property is *has matrix*, and the object is *Matrix1*.

RDF is based on the idea of identifying things using *Uniform Resource Identifiers* (URIs). A URI is a string which identifies a resource in an unique way, such as images, documents, files, or any resource which can be accessed through the network. For instance, the URI <http://ImageExample.com♯Image1> can be used to identify the resource *An image*, while the URI <http://Image.org♯hasmatrix> can be used to identify the resource *has matrix*. Sometimes, objects can be represented using values (denoted as *RDF Literals*) instead of URIs. In the statement *The matrix Matrix1 X-dimension is 20*, the resource *20* can be represented as the RDF Literal *20*, which corresponds to an integer.

In order to simplify and enhace the writing of documents, RDF also allows the use of prefixes. For example, the URI <http://ImageExample.com♯ > can be assigned a prefix (@prefix ex: <http://ImageExample.com♯ >). Then, *ex* : *Image*1 is a shortcut for the URI <http://Image Example.com♯Image1>.

RDF statements are composed of three components, which are required to be written in an ordered way `(subject, predicate, value)`. In RDF terminology, this is called an *RDF triple*. For instance, the statement *A given Image has a Matrix called Matrix1* can be represented with the RDF triple ⟨ex:Image1, im:hasMatrix, ex:Matrix1⟩. When several statements need to be linked, RDF uses sets of RDF triples, called *RDF graphs*. For example, in order to link the statements *An image has a matrix called Matrix1* and *The Matrix Matrix1 X-dimension is 20*, the following RDF graph composed of two RDF triples could be used: {⟨ex:Image1, im:hasMatrix, ex:Matrix1⟩, ⟨ ex:Matrix1, Im:hasDimx, 20⟩}.

The *RDF Schema* (RDFS) [27] is a language to represent resources (as is the case of RDF) which can also deal with rules over classes, subclasses and properties of classes. It can be used to represent the membership to a class (rdf:type), subsumption between classes (rdfs:subClassOf), subsumption between properties (rdfs:subPropertyOf), the domain (rdfs:domain) and range (rdfs:range) of properties, the meta-class of classes (rdfs:Class), the meta-class of properties (rdf:Property), etc. Rules allow deducing new data from some given data. For example, from a rule satating that class A is a subclass of class B and that class B is a subclass of class C can be deduced that class A is a subclass of class C. A class in RDF (or RDF Schema) corresponds to the generic concept of a type or category, somewhat

like the notion of a class in object-oriented programming languages.

The Web Ontology Language (OWL) [28] includes more complex rules for describing classes based on allowed values for properties. OWL improves the machine interpretability of Web content supported by RDF and RDF Schema by providing an additional vocabulary with a formal semantics. For example, that two classes are equal (owl:sameAs), equivalent (owl:equivalentClass) or have equivalent properties (owl:equivalentProperty). An RDF graph that uses the OWL vocabulary to define classes and properties is generally called and *ontology*.

Given two RDF graphs g and g', it is said that g simply *entails* g' ($g \models_{RDF} g'$) when each simple interpretation that satisfies g also satisfies g' [29] (the semantic information of g is sufficient to satisfy the semantic information of g'). For example, it is easy to see that the RDF graph g = {⟨a, b, c⟩, ⟨d, c, e⟩, ⟨f, h, i⟩} simply entails the RDF graph g' = {⟨a, b, c⟩, ⟨d, c, e⟩} being a, b, c, d, e, f, h and i different URIs.

In order to perform queries about RDF Data, *RDF graph patterns* are used. An *RDF graph pattern* is a set of *RDF triple patterns*. An RDF triple pattern, like an RDF triple, contains three fields: ⟨subject, predicate, object⟩. The difference is that they can also contain variables (belonging to a given set of variables) in any field. For example, the RDF triple pattern ⟨Matrix1, Im:hasDimx, ?Dimx⟩ can be used to recover the X-dimension of Matrix1. The RDF triple pattern ⟨?s ?p ?o⟩ can recover any RDF triple. Query languages for semantic data generally assume that implicit triples have been inferred and added to the database. SPARQL is the most used language for querying RDF Data. Query languages over semantic Web-based databases have the same aims as SQL on relational databases. The most well-known query language for querying RDF graphs is SPARQL [30]. It reuses the SELECT FROM WHERE shape of SQL queries (used for querying relational data) including RDF graph patterns in the WHERE clause. The atomic graph patterns are RDF triple patterns, and the composing binary operations are AND (join), UNION (set union) and OPTIONAL (left join). The new version of SPARQL (SPARQL 1.1 [31]) extends the previous one including EXISTS, NOT EXISTS, MINUS, path expressions, GROUP BY, HAVING and aggregation functions (Flatten, Count, Sum, Avg, Min, Max, etc.).

A. DESCRIBING INPUTS, OUTPUTS, PRECONDITIONS AND POSTCONDITIONS

In this paper, RDF graph patterns are used to represent the set of inputs and outputs of each task composing the workflow. An RDF graph pattern is used for each input parameter, so that any actual data entailing the graph pattern could be used as an input value. Figure 3 depicts an intuitive view of task *align_warp* inside the scientific workflow specification of the First Provenance Challenge [24] shown in Figure 1. In the figure, RDF graphs are shown using a graphical notation according to the U-RDF-PN ontology definition.

?*ai* and ?*ri* are RDF-graphs that input actual parameters (data provided at task invocation time) must entail. They impose a kind of structural conditions for the provided graph data: any RDF graphs entailing graph pattern ?*ai* and ?*ri*, respectively, can be used to feed the image to be treated in the `align_warp` service. On the other hand, the *Precondition* imposes some constraints on the values provided to fulfill input graph pattern variables. In this case, the following conditions are imposed: the images must have either "jpg" or "tiff" format, both the image to be processed and the reference image must have the same format and, finally, the reference image must effectively be the one the input ?*ai* parameter is waiting for.

Analogously, the output graph pattern ?*warp* establishes the structure of the data returned by the task execution call, while *Postcondition* states how values in the output graph will be obtained in terms of the actual parameters. In this case, the postcondition establishes that the result stores the name of the input image as well as the name of the reference image. The last part establishes that as a result of the execution, the warped image will have some new data computed from the input parameters (the input image, `ai1`, and the reference image, `ri`).

IV. A CLASS OF SCIENTIFIC WORKFLOW MODELS ENHANCED WITH SEMANTIC INFORMATION

In this paper, Petri nets are used for the specification of scientific workflows. Petri nets are a technology widely used in the world of workflows [32]–[34] and also in the case of scientific workflows [35], [36]. In order to be able to incorporate semantic information to Petri net models, we developed the class of Unary-RDF-Petri nets, U-RDF-PN [37]. This class belongs to the family of hight level Petri nets. U-RDF-PN share the notions of place, arc, transition and marking. Semantics are then incorporated in the *tokens* of the nets (tokens are RDF graphs with information related to the object the token refers to), in arc inscriptions (which are RDF graph-patterns) and, finally, in transitions (incorporating preconditions, postconditions and guards related to the transition input and output arc inscriptions).

As usual in Petri nets, a system state is modeled by means of the net marking while system evolutions are represented as the enabling and firing of transitions. Figure 4 sketches the Petri net view of the task invocation `align_warp` in previous Figure 1. Figure 4-a) depicts formal and actual parameters, whereas Figure 4-b) shows how token *warp*1 will appear in place *p*11 once transition *align_warp* is fired. As it is shown, tasks are modeled by means of transitions as follows:

- Data feeding the task will correspond to tokens (RDF graphs) in the transition input places (*p*00 and *p*01). On the other hand, data produced by the task (output data) will correspond to the RDF graphs put by the transition firing in transition output places. In the case of the `align_warp` task input data corresponds with the

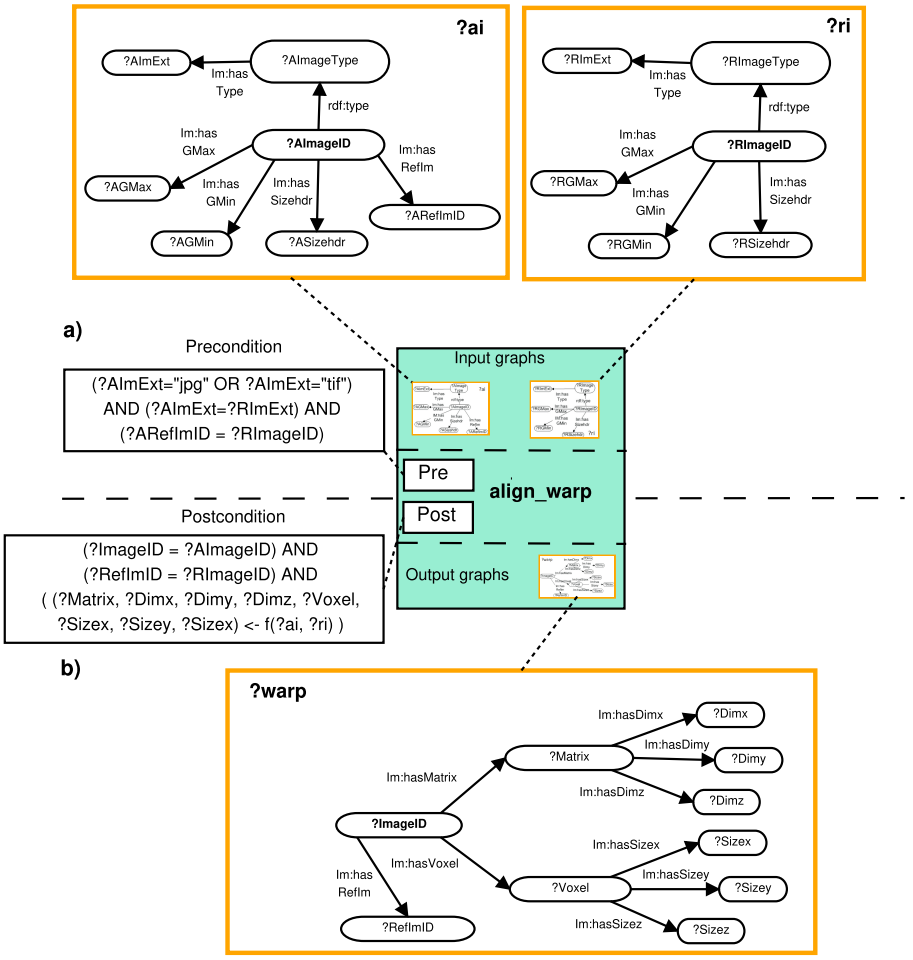

FIGURE 3. Intuitive description of task `align_warp` in Figure 1. a) Input graphs and preconditions. b) Output token (RDF graph) and postcondition.

RDF graphs ai1 and r1 while the output data corresponds to the RDF graph warp1 (Figure 4).

- Task inputs will be modeled by means of the RDF graph patterns in the transition input arcs. At a given system-state, a token in an input place entailing the RDF graph pattern in the corresponding arc is a candidate for firing the transition. In our case the RDF graph patterns ?ai and ?ri (Figure 4) model the inputs of the task *align_warp*. Tokens ai1 and r1 entail the RDF graph patterns ?ai and ?ri, respectively, and they are candidates for firing the transition.
- As it was stated, a transition can have a guard attached. The guard is nothing else but a boolean expression involving URIs, literals and input variables (variables used in RDF graph patterns attached to the input arcs). As usual in high level Petri nets, that guards can be used to implement the preconditions of the task. Among the possible graph candidates, only those satisfying the guard can be used to enable the transition.
- When can the transition fire? Candidate tokens must be found for every input arc so that a binding is possible (which means that the same variable in different input arcs must correspond to the same value in the different input candidate tokens) and so that the guard attached to the transition is made true for the chosen values. In our example, transition *align_warp* can fire because it is possible to find a binding assigning values (RDF Literals or URIS) to the variables of the RDF graph patterns ?ai and ?ri (this binding assigns for example the URI ex:Image1 to the variable ?ImageID or the RDF Literal 22 to the variable ?Dimx) and the precondition (?AImExt = "jpg"OR ?AImExt = "tif") AND (?AImExt = ?RImExt) AND (?ARefImID = ?RImageID) is true (the extension of both figures is "jpg" and ex:RefImage is the reference image of ex:Image1).
- The RDF graph pattern attached to the output arcs represent task outputs, while the postcondition mapping associates to a transition its corresponding postcondition, establishing the relation between the input and output task values. In our example the postcondition (?imageID = ?Aimage) AND (?RefImaID = ?RImageID) AND (?Matrix, ?Dimx, ?Dimy, ?Dimz, ?Voxel,

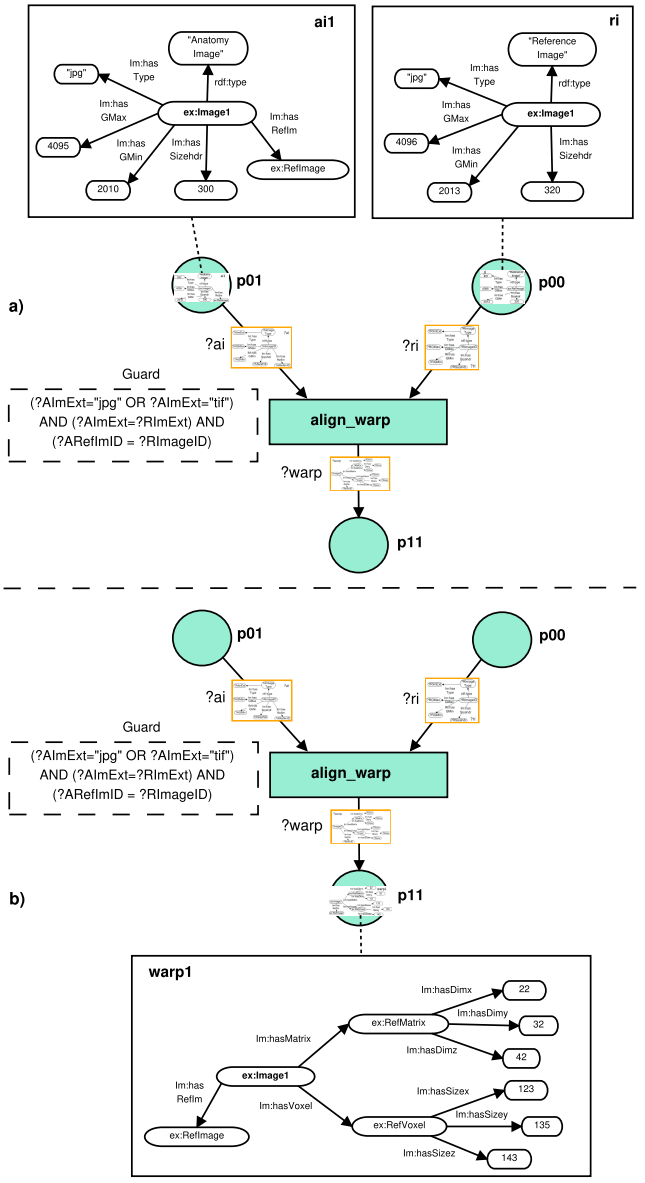

FIGURE 4. A Petri net view of the task invocation `align_warp` in Figure 1.

?Sizex, ?Sizey, ?Sizez) ← f(?ai,?ri) establishes a relation between the inputs (expressed with the RDF graphs ai1 and r1) and the outputs (expressed with the RDF graph warp1). It gives values to the variables ?imageID and ?RefImaID using information of the input tokens (a1 and ri) and it also gives values for the rest of variables of the RDF graph pattern ?warp using function f. In the example sketched in Figure 4, in case of transition `align_warp` fires choosing input tokens `ai1` and `ri`, that tokens will be removed from places p01 and p00, respectively, and token warp1 will be put in place p11.

The previous description deals with the way a task can be modeled in a U-RDF-PN. However, as Petri net models, they also include in a natural way the patterns used in (scientific) workflows: parallel split, join, sequential composition, etc.

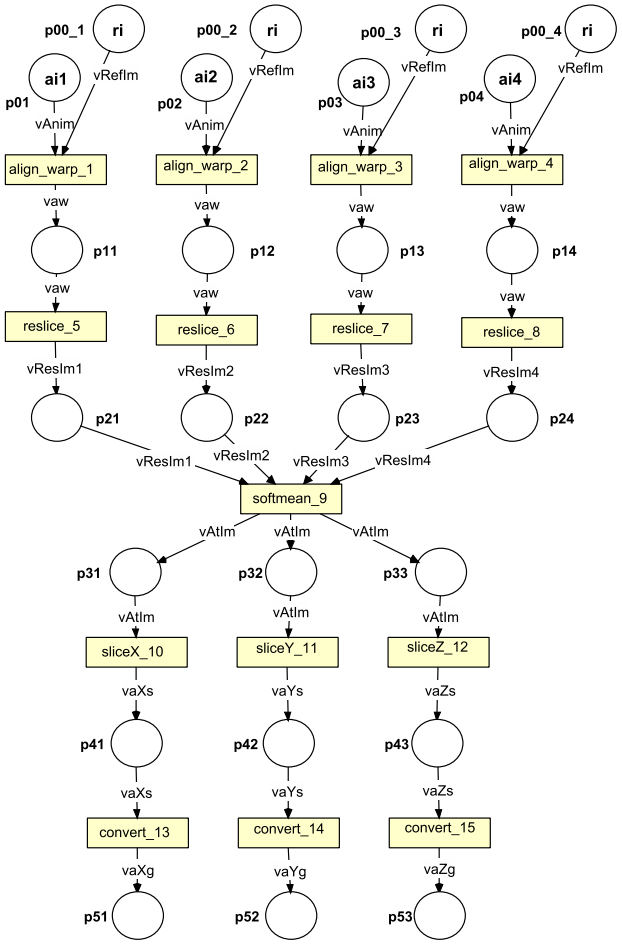

FIGURE 5. U-RDF-PN modeling the first provenance challenge workflow.

As usual with Petri net based formalisms, the system semantics are defined around four main concepts: `system marking`, which defines the distribution of tokens in places, modeling the state of the system; `transition enabling`, which determines the conditions under which a system evolution can occur; `transition firing`, which gives the effective state change; and `reachability graph`, corresponding to the set of system states and state transitions that can be reached by the system from a given initial system state. The Petri net in Figure 5 shows the U-RDF-PN model corresponding to the workflow shown in our first example, depicted in Figure 1.

A. THE REACHABILITY GRAPH

For a given Petri net, the *reachability graph* is the graph containing the set of reachable states (markings) as nodes and an arc joining two states m_1 and m_2 if m_2 is reachable from m_1 by firing a transition. Usually, since the transition will be fired for a concrete binding, the binding itself can be associated with the arc.

The generation of the reachability graph is based on the classical algorithm used for computation of the reachability graph in Petri nets [38]. This version of the algorithm

has been modified to consider RDF annotations. The input is an RDF graph representing the U-RDF-PN system. The algorithm is quite simple. In the first step, the reachability graph is empty, and the initial marking is the only marking on a marking stack. Then an iterative process is applied as follows: take the marking in the top of the stack; compute the possible firings from that marking, as well as the marking reached for each one of the possible firings; if any of these new markings are not in the until-now computed graph, add such marking and state transition to the graph, and also push this marking into the stack. The process terminates when the stack is empty. The method relies on two main functions. The first one is the function that, for a given marking, computes the set of enabled transition firings, as well as the markings reached in case of such firings. The second one is the function that, given two markings, determines whether they are equivalent or not. Let us now concentrate on these aspects. The functions involved in the reachability graph computation must work with RDF annotations: the function looking for enabled transitions, the function for firing a transition and, finally, the function checking for the equivalence of two markings. The implementation of these RDF-based functions is our main contribution regarding other implementations of the classical algorithm for the computation of the reachability graph of a Petri net. Both checking for enabled transitions and equivalence between two markings have been programmed as SPARQL queries (a SELECT and an ASK query, respectively), whereas the firing of a transition has been implemented using some functionalities provided by the RDF database used for storing the input system. Both solutions are easily integrated into our Java-based algorithm implementation.

As an example let us show how it can be checked whether two markings are equivalent. Let *rg* be a reachability graph and *S* be a marking. The following SPARQL ASK query checks if a marking exists in *rg* that is equivalent to *S*:

```
1. PREFIX rg: <http://rg.org/>
2. ASK {
3.    ⋀_{i=1}^{k} GRAPH rg : g_i {
4.       rg : g_i  rg : IsContainedInS  ?S .
5.       rg : g_i  rg : IsContainedInP  rg : q_i .
6.       rg : t_{g_1} . . . . rg : t_{g_{i_k}} . }
7. OPTIONAL {?newGraph rdf :type  rg :RDFGraph .
8. FILTER (⋀_{i=1}^{k} ?newGraph != rg : g_i) } .
9. FILTER (!bound(?newGraph)) } ,
```

where

- S is the marking to be checked for the equivalence ($\langle rg : S, rdf : type, rg : State\rangle$),
- $\{g_1,...,g_k\}$ are the RDF graphs contained in marking S. ($\langle rg : g_i, rdf : type, rg : RDFGraph\rangle$, $\langle rg : g_i, rg : IsContainedInS, rg : S\rangle$ $(i \in \{1..k\})$,
- $g_i = \{t_{g_1},...,t_{g_{i_n}}\}$ is the set of RDF Triples of g_i ($\langle rg : t_{g_j}, rdf : type, rg : RDFTriple\rangle$, $\langle rg : g_i, rg : hasT, rg : t_{g_j}\rangle$ $(j \in 1..i_n)$).

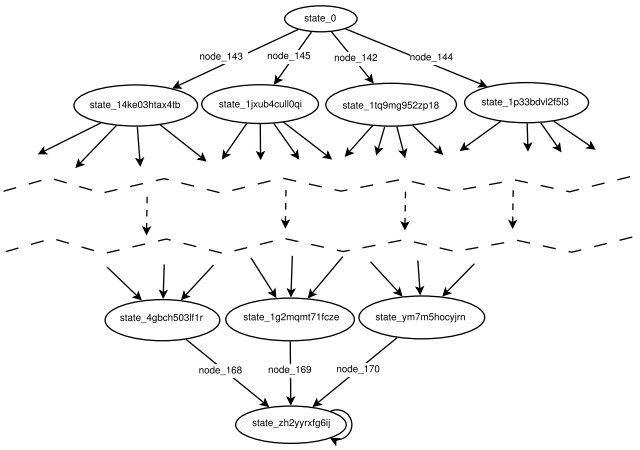

FIGURE 6. A partial view of the reachability graph of the model in Figure 5.

- and q_i is the place that contains graphs g_i in the marking S ($\langle rg : q_i, rdf : type, rg : Place\rangle$, $\langle rg : g_i, rg : IsContainedInP, rg : q_i\rangle$ $(i \in \{1..k\})$.

From a more intuitive point of view, the ASK query is executed on the RDF database where the reachability graph *rg* is stored. From marking *S*, Lines 3-6 define the set of RDF graphs the target marking must contain. If a marking is found, then it entails *S*. In order to guarantee the equivalence between the found marking and *S*, lines 7-9 require *S* also entailing the found marking and, therefore, both markings are composed of sets of equivalent RDF graphs.

Once the algorithm execution terminates, the obtained graph contains the set of possible states the system can reach, as well as the set of possible transition firings, which correspond to the set of possible system evolutions. Figure 6 shows an excerpt of the reachability graph of the model in Figure 5 (the whole graph contains 108 states). Let us concentrate on the first possible evolution. Node labeled as `state_0` corresponds to the initial system configuration, in which graphs `ai1` to `ai4` are in places `p01` to `p04`, respectively, and a graph `r1` appears in each `p00_`$\star$ place (these places are equivalent to place `p00` in Figure 4). From this state four transitions could be fired: `align_warp_1` to `align_warp_4`, corresponding each firing to the `align_warp` service invocation with the different given images. In the reachability graph, these four possibilities correspond to the four reachable state appearing on top of Figure 6, with the corresponding arcs (node_143, node_145, node_142 and node_144). The central part of the figure corresponds to the system state in which, once the four input images have been *warped* and *resliced*, state named `state_18iegssrtgl1va` corresponds to the marking in which there is an RDF graph describing the warped and resliced images in each `p21,p22,p23,p24` places, transition `softmean_9` can be fired, putting the same token in each `p31,p32,p33` places, being `state_bxsh6519ej74` the new state.

Bottom part of the figure corresponds to the final part of the execution in which all the images have been processed and the X, Y and Z views have been obtained.

V. EXPLOITING THE MODEL: BEHAVIORAL ANALYSIS

As previously stated, the reachability graph is the representation of the possible system executions. In fact, it can be viewed as an automaton recognizing the set of possible transition firings as the language. The behavior corresponds to the properties (all the/some of the) executions verify. An execution corresponds to a possible interleaving of actions of the involved processes.

This notion of *execution* introduces a new dimension to the logical truth of propositions. In the usual mathematical logic, for a given formula, an interpretation assigns values to atomic variables, making the proposition to be either *true* or *false*. In the case of executions, the value of variables can change (due to the value assignments) and then the truth of the proposition can also change along time. To deal with this, *temporal logic* was introduced as a formal system adding operators to manage time. This is possible because in the interleaved executed actions one action comes after the other, and so a notion of time can be considered. This *time* is not a physical time, but rather a notion of present, past and future related to the current, past and future system states.

Together with the classical logical operators (AND, OR, etc.) two new kinds of operators are considered, A and E. At a given state s, A (E) formula refer to something verified along all the execution paths (at least one path) starting at s.

Among the different versions of temporal logic (see [39], [40] for a good review), we have chosen *Computation Tree Logic* (CTL). CTL formulas can begin with one of the AX, EX, AG, EG, AU, EU, AF, EF connectives. The first operator is either A or E. At a given state s, A (E) formula refer to something verified along all the paths (at least one path) starting at s. The second operator is one of the set $\{X, F, G, U\}$, meaning ''neXt state'', ''some Future state'', ''all future states (Globally)'' and Until, respectively. Symbols X,F,G,U cannot occur without being preceded by an A or an E; similarly, every A or E has to have one of X,F,G,U after it.

Temporal logic formula can be used to specify properties of the system behavior. For instance: 1) Is the value of variable x always positive? (AGp formula, where p corresponds to x being positive) 2) Can I ensure that the workflow will always terminate properly? (AFp being p the predicate establishing proper termination) 3) Can I ensure that the workflow will sometimes terminate properly? (EFp being p as in the previous case) 4) Is it possible to process the input set of images when they have different formats, sizes, etc.? ($q \wedge EFp$ where q states that the input images have different formats while p states that the images have been correctly processed).

Figure 7 depicts the main operators in an intuitive way. The root node corresponds to the state where proposition p is being studied, while black nodes represent states that must verify p in order the temporal logic formula labeling the corresponding figure be true. In the case of *until* formula (the two formula at the right of the figure) non-bordered nodes correspond to states that should verify the q part of the formula, while black nodes correspond to those that must verify p.

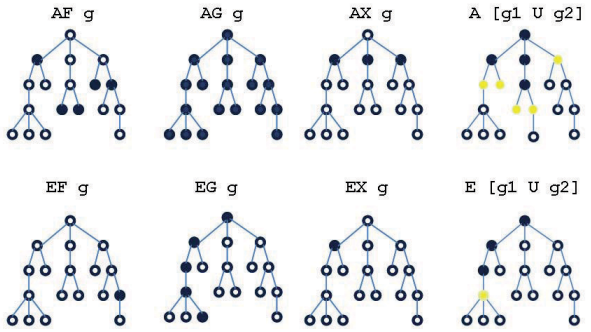

FIGURE 7. Abstract representation of the main CTL temporal operators.

The second important concept is *model checking*. A model checker takes a model of the system (the reachability graph in our case),[2] a temporal logic formula stating the property we want to verify and a state of the model as inputs. Then, the model checker determines wether the formula is verified by the model at the given state. In our case, the considered state is always the initial state. Let us describe the process in more detail in the following sections.

A. AN ENVIRONMENT FOR MODEL-CHECKING

Figure 8 depicts the practical point of view of the approach presented in this paper, the COMBAS framework [41]. This framework has been updated over time and integrates a set of tools for the generation of U-RDF-PN models, the corresponding reachability graph and its corresponding RDF Annotated Kripke Structure [23]. COMBAS also allows the creation and edition of queries and CTL formulae and the execution of the model checking process.

From the user's perspective, the input of the model checking environment is composed of a scientific workflow described as a U-RDF-PN using the PNML [42] standard[3] and also its initial markings. A set of parsing tools has been developed and integrated in COMBAS, allowing to use different input systems, such as a process logs or a provenance trace, for example, which are converted to the U-RDF-PN description for their processing.

The *CTL formula* is also provided as an input of the system. We should not forget that final users will usually be scientists, not computer scientists. This means that maybe we are fac-

[2]Actually, it is a slightly modified reachability graph since the required input is a Kripke structure (namely RDF-KS), which must be left-total: every state must have at least one successor state. So, for those states corresponding to total deadlocks a virtual dead state is added, which closes over himself. This is just a technical adaption. This is the case of the self-loop in the last state of the graph sketched in Figure 6.

[3]Petri Net Markup Language (ISO/IEC 15909), an XML based description for Petri nets.

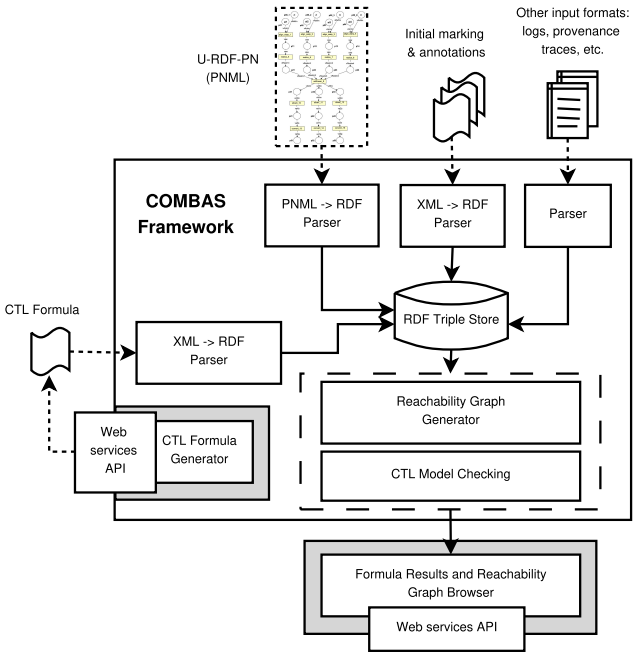

FIGURE 8. Architecture of the COMBAS model-checking framework.

ing with people that require some more friendly interfaces than just a text editor for writing the formula. Some efforts have been done in these directions, as shown in [43]. In the COMBAS environment, the *CTL Formula Generator* is a graphical Web-based tool that enables to easily build CTL formulae.

During the model checking process, an RDF database is used (*RDF Triple Store* in Figure 8). The COMBAS framework allows using several different RDF databases, such as the *AllegroGraph RDFStore* database or the *Virtuosso RDF Store*, for example. The only requirement is that the RDF store must allow to be accessed through a SPARQL interface. In this paper the Virtuosso RDF Store has been used.

As a result from the processing, the truth about the verification of the formula is obtained. Moreover, a graph depicting the reachability graph states can be browsed using a graphical Web-based enabled interface (*Formula results and reachability graph browser* in Figure 8). Doing so, it is possible to find the specific situations in which a predicate violates some wanted condition, having a better insight of the workflow behavior, and making easier the workflow improvement.

All components in COMBAS expose an easy, flexible and usable interface, and the complexity of the graph generation, storage in the semantic triple store and verification processes are hidden from the user's perspective.

Internally, the generation of the reachability graph is based on the classical algorithm used for computation of the reachability graph in Petri nets described in subsection IV-A. A parallel and scalable approach for the parallel computation of the reachability graph in COMBAS can be found in [44].

The implementation of our model checker is based on an adaptation of the labeling algorithm proposed in [40]. The inputs are an RDF-KS and an RDF CTL formula. Both inputs are stored into the RDF database following the corresponding RDF schemas. Basically, the algorithm computes the set of states of the input system $\mathcal{M}$ that satisfy the given CTL formula ϕ. This process consists of three steps. Initially, the formula ϕ is translated into an equivalent formula in terms of the connectives AF, EU, EX, $\top$, $\wedge$ and $\neg$. The equivalence rules applied are defined as part of the labeling algorithm. Secondly, the states of $\mathcal{M}$ satisfying subformulas of ϕ (ψ) are labeled, starting with the smallest subformulas and finishing with the original formula. Finally, the algorithm returns the states labeled with ϕ.

One of the relevant modifications of the algorithm is the RDF encoding of the input parameters, $\mathcal{M}$ and ϕ. This encoding makes possible to implement the `EntailsRDF` and `EntailsBGP` in steps 2-b) and 2-c) of the procedure as SPARQL queries. Let us take a deeper look at the `EntailsRDF` function. Let us suppose that the RDF database $M.rdf$ stores the RDF-KS $\mathcal{M}$. The RDF Schemas used to represent $\mathcal{M}$ and ϕ are denoted as *ks* and *ctl*, respectively. In this case the formula is an RDF graph defined by ($\langle ctl:g, rdf:type, ctl:RDFGraph\rangle$), where t_{g_1},..,t_{g_n} are the set of RDF triples of g ($\langle ctl:t_{g_j}, rdf:type, ctl:RDFTriple\rangle$, $\langle ctl:g_i, ctl:hasT, ctl:t_{g_j}\rangle$ ($j \in 1..n$)).

In order to calculate the set of states of the reachability graph $\mathcal{M}$ satisfying formula ϕ, the WHERE clause (Lines 4-6) imposes the resulting states (Line 3) to satisfy $g' \models_{RDF} g$, for some $g' \in L(s)$ (in our RDF-KS definition, for a state s, $L(s)$ is the set of graphs that represent the state).

```
1. PREFIX ks: <http://ks.org/>
2. PREFIX ctl: <http://ctl.org/>
3. SELECT ?S FROM M.rdf
4. WHERE {GRAPH ctl:g { ctl:g ks:IsContainedInS
                        ?S .
5.                      ?S rdf:Type ks:state .
6.                      ctl:t_g1 .... ctl:t_gn . } }
```

Finally, it must be said that the core of the algorithm (the translation and the breaking down of the input formula, the labeling of states, etc.) has been programmed using the Java programming language.

Let us now express and check some interesting properties of the example depicted in Section II, and which can be solved using the COMBAS framework.

Can I ensure that the workflow will always terminate properly? One should first model how this property could be described in terms of the model. In this case it is quite clear. A proper termination would correspond to the case in which transitions `convert_13`, `convert_14`, `convert_15` have fired, putting the corresponding tokens in places `p51`, `p52`, `p53`, respectively. The token in place `p51`, for instance, must also have come firing transition `convert_13`, which means that it must entail graph pattern `vaXg`. Another important constraint for correctness is that the set of images in the last state must all have the reference image, and correspond to the same batch of images (in this

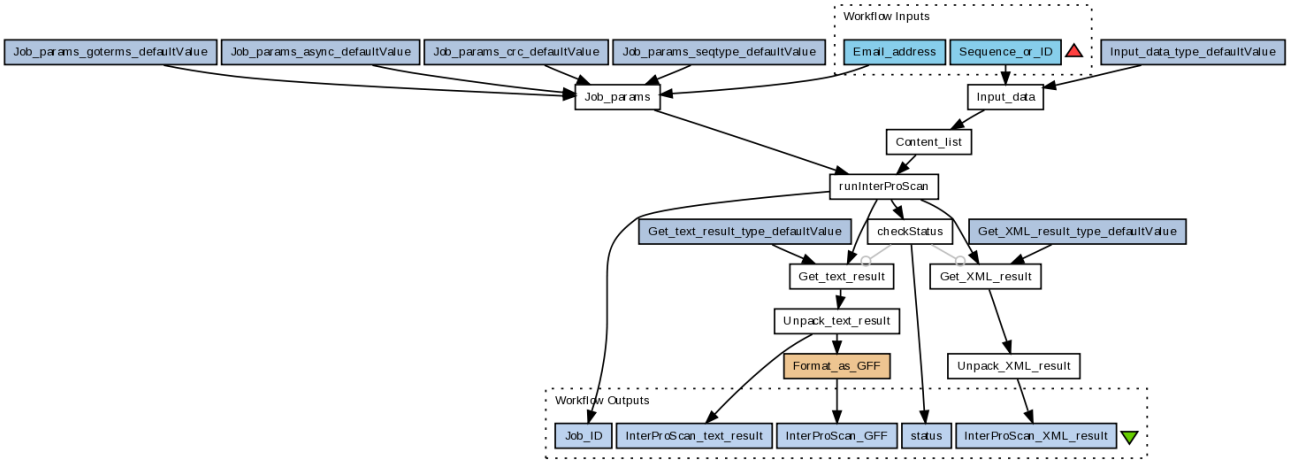

FIGURE 9. Workflow for the InterProScan analysis of a protein sequence.

case we are just considering a unique batch, but in a general case one could think about different batches, which require preventing images from different batches to be mixed). So, a possible formula to be checked for that property is [$\mathcal{M}$,$S1$ $\models AGEF(p51(vaXg) \wedge p52(vaYg) \wedge p53(vaZg))$]. The formula holds because for each branch in the RDF-KS it is possible to reach a state with an RDF graph in places $p51$, $p52$ and $p53$ satisfying the RDF graph patterns *vaXg*, *vaYg* and *vaZg*.

Can a graphical atlas image be obtained if one of the anatomy images has not been resliced? In our model this implies to find a state `s` that contains an RDF graph in places $p51$, $p52$ and $p53$ satisfying the RDF graph patterns *vaXg*, *vaYg* and *vaZg* and an RDF graph satisfying the RDF graph pattern *vaw* in places $p11$, $p12$, $p13$ or $p14$. This property could be checked with the formula [$\mathcal{M}$,$S1$ $\models EF(p51(vaXg) \wedge p52(vaYg) \wedge p53(vaZg) \wedge (p11(vaw) \vee p12(vaw) \vee p13(vaw) \vee p14(vaw))$]. In this case the formula does not hold because it is not posible to reach an state in which the workflow has obtained a graphical atlas image if there are a token in places $p11$, $p12$, $p13$ or $p14$. In this situation transition `softmean_9` could not have been fired.

VI. INTERPROSCAN ANALYSIS OF A PROTEIN SEQUENCE

In Section II, the First Provenance Challenge problem, which constitutes a traditional environment for scientific computing, was used to introduce the applicability of the solution proposed in this paper. As it was shown, a current trend in scientific computing is to look for processing tools and services over the network, integrating into our workflows procedures and services deployed by external entities.

Let us now concentrate on this kind of solutions, by showing how to analyze a protein sequence using the EMBL-EBI's InterProScan Web services[4] from the The European Bioinformatics Institute.[5] The processing is based on the InterProScan workflow, an integration workflow for the signature-recognition methods in InterPro [45] and depicted in Figure 9.[6] In this case, we have updated the InterProScan workflow to use the latest EMBL-EBI's InterProScan Web services.

The workflow's input is composed of the sequence to be processed and a user email address for notification purposes. There are a few more parameters required for the analysis, set to their default values for this example. Starting with this input, a protein sequence is searched inside a set of protein families and domain signature databases integrated in InterPro.[7] As a result, a set of matches are properly formatted and returned. These matches are also annotated with the corresponding InterPro and GO term assignments.

Figure 10 depicts the proposed Petri net model of the workflow in Figure 9, using the U-RDF-PN formalism. All the data flowing through the workflow have been semantically annotated using instances of the PRO Protein Ontology [46]. In order to describe the `Sequence_or_ID` input of the workflow (one of the two mandatory inputs) the Protein Complex Concept of the protein ontology has been chosen [47].

A protein complex is a group of two or more associated proteins. Its subconcepts *Structure*, *StructuralDomains* and *FunctionalDomains* provide complete understanding of the sequence, structure and functional interactions. In such case, we can use `po` as the prefix to denote the protein ontology. Therefore, the RDF triple `:_Protein Complex rdf:type po:ProteinComplex` can be included in the `Sequence_or_ID` RDF graph.

For the sake of clarity, the input sequence has been bound to four proteins in this example. Transitions Input_data_1 to Input_data_4 depict the process to obtain the four input proteins, which will then compose a ProteinComplex structure.

In order to execute the `runInterProScan` activity, two different Web services, `runInterProScan1` and `runInterProScan2`, are available in a repository.

[4]http://www.ebi.ac.uk/Tools/webservices/
[5]http://www.ebi.ac.uk
[6]http://www.myexperiment.org/workflows/814.html
[7]http://www.ebi.ac.uk/interpro/

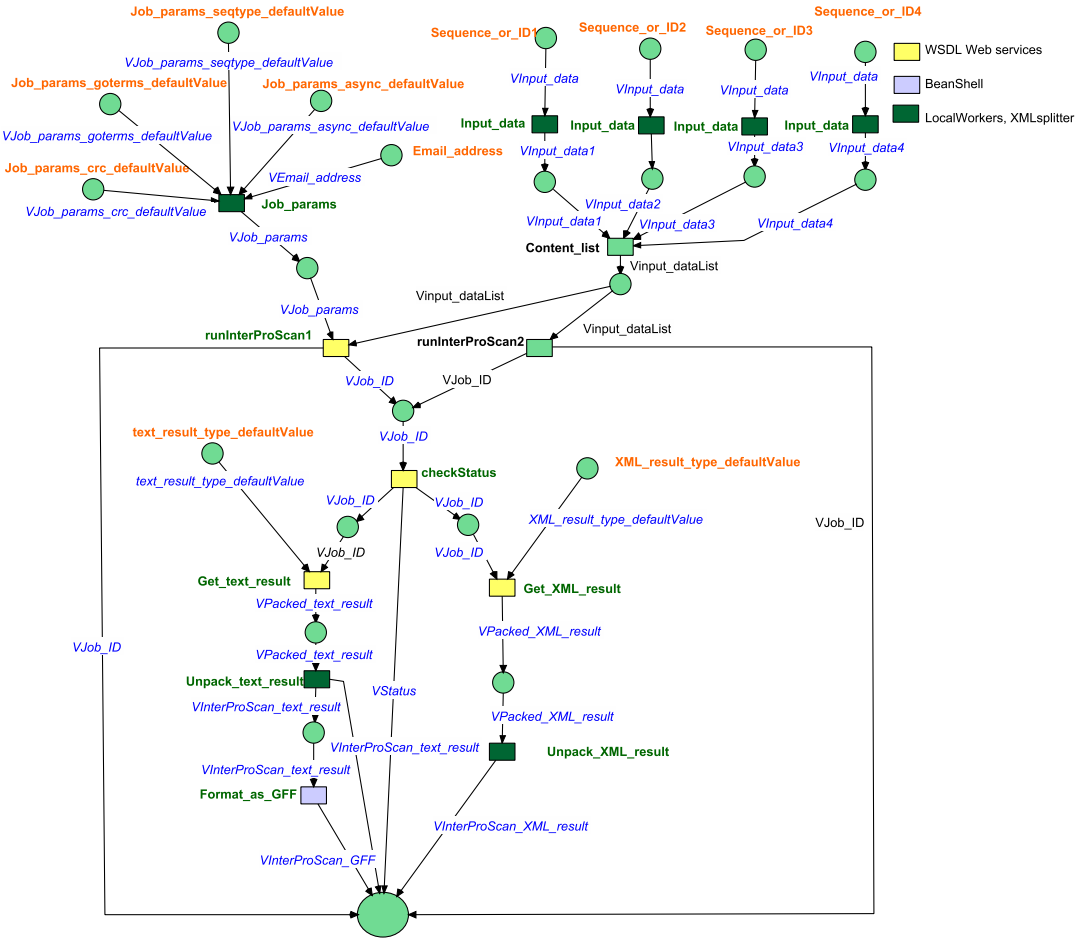

FIGURE 10. U-RDF-PN modeling of the workflow for the analysis of a protein sequence presented in Figure 9.

Both services implement EMBL-EBI's InterProScan Web services, and they are able to carry out the protein analysis, which is the more expensive part of the experiment. Both services can also carry out the analysis using the protein crystallographic data of the sequence (represented in the ontology with the Unit Cell concept). To model this, the RDF graph pattern of the input arc of the transitions include the RDF triple pattern `:_ProteinComplex rdf:type ?rdftype`, while the transitions include a guard imposing the type to be a po:UnitCell concept (?rdftype = po:UnitCell). In addition, the RDF graph representing the information of the input data includes the RDF triple `:_ProteinComplex rdf:type po:ProteinComplex`. We can see that it is possible to execute this service because po:ProteinComplex is a super concept of po:UnitCell, and the guard is then evaluated to true.

However, let us suppose that the first service, `runInterProScan 1`, is able to return information about the functional properties of the interaction while the second one, `runInterProScan 2`, is only able to return information about the structure. Functional interactions of proteins are represented with the Source Cell concept of the protein ontology. The RDF graph representing the information of the output data of transition `runInterProScan1` will include the RDF triple `:_ProteinComplex rdf:type po:SourceCell`. However, the RDF graph representing the information of the output data of transition `runInterProScan2` will include the RDF triple `:_ProteinComplex rdf:type po:UnitCell`. In this last case, the output data can also contain different parameters describing crystallographic data: `:_ProteinComplex po:A :_A` (a in angstroms), `:_ProteinComplex po:B :_B` (b in angstroms), `:_ProteinComplex po:C :_C` (c in angstroms), `:_ProteinComplex po:Alpha :_Alpha` (alpha in angstroms), `:_ProteinComplex po:Beta :_Beta` (beta in angstroms), `:_ProteinComplex po:Gamma :_Gamma` (gamma in angstroms), `:_ProteinComplex po:z :_z` (Z value), etc.

On the other hand, let us suppose that in the Web service repository all the Web services that can be invoked for the `checkStatus` activity (transition *checkStatuts* in Figure 10) need information about the functional interactions of the sequence proteins in order to be executed.

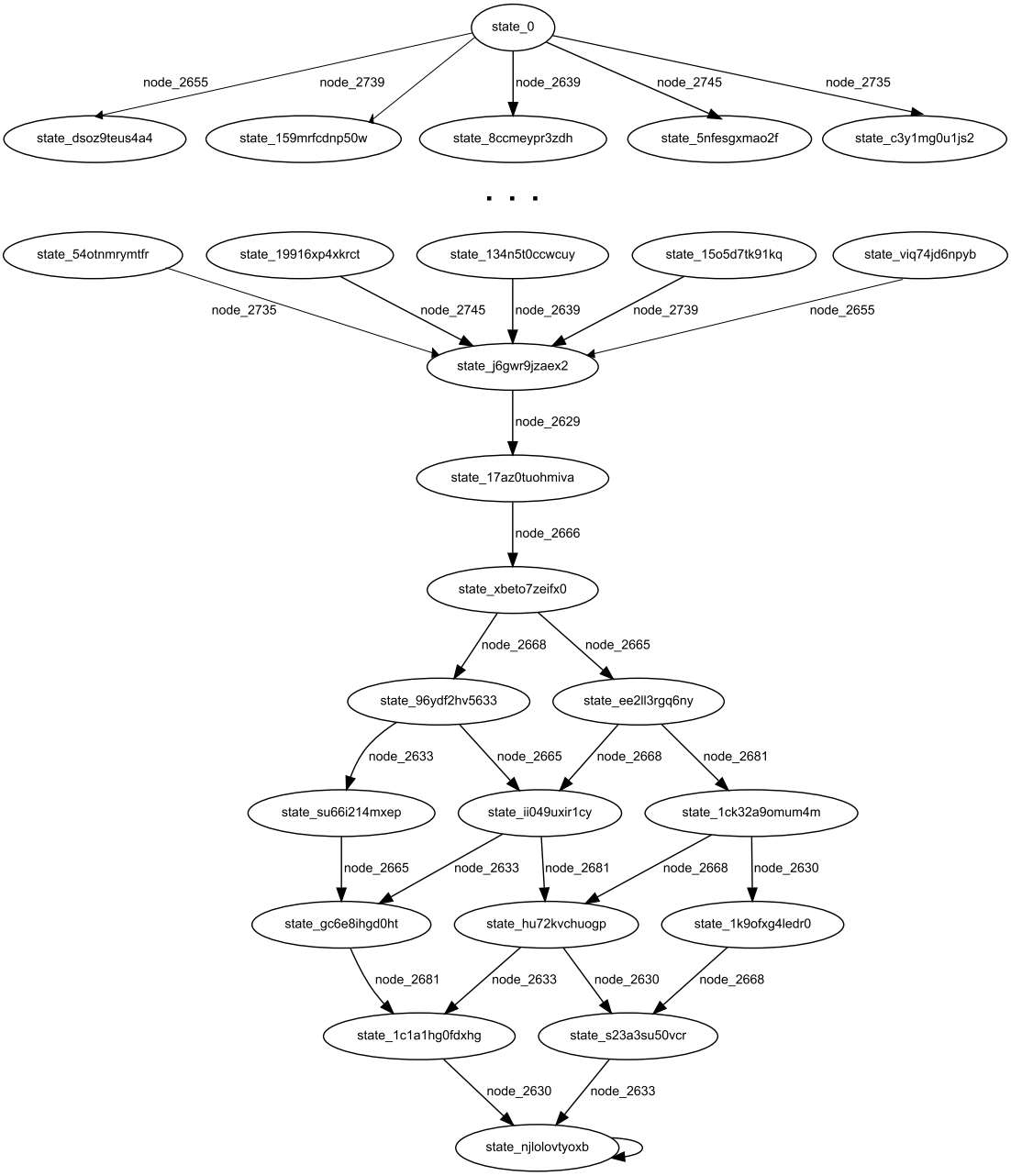

FIGURE 11. A partial view of the reachability graph of the model in Figure 10.

To model this, the RDF graph pattern of the input arc of transition `runInterProScan` should include the RDF triple pattern `:_ProteinComplex rdf:type ?rdftype` as in the case of transitions `runInterProScan`. In this case the transition guard should express that the type should be a po:SourceCell concept (?rdftype = po:SourceCell). In this situation, the token in the input place of transition `checkStatus` can reach that place by either firing transition `runInterProScan 1` or `runInterProScan 2`. In the first case, `checkStatus` could be fired after, but not in the second case. This last case is due to the fact that the information in the token contains the crystallographic data of the sequence, but the service needs the information about functional interactions as input.

If we built the reachability graph of this system we could see that there are not states including the `InterProScan_GFF`, `VInterProScan_text_result`, `InterProScan_XML_result` and `Status` outputs when transition `runInterProScan 2` is fired. The experiment execution will fail. This can be easily asked using the temporal logic formula: AG(p_scaned(structureInformation) $\Rightarrow$ EF(p_scanedInterProScan_GFF $\vee$ VInterProScan_text_result $\vee$ InterProScan_XML_result $\vee$ Status)).

Figure 11 depicts a partial view of the generated reachability graph. As it can be seen, there is a clear separation between the processing up to the `checkStatus` transition firing in Figure 9 (places between `state_0` and `state_17az0tuohmiva` in the reachability graph), in which there is a state explosion, and the rest of the workflow (which is simpler). The full reachability graph contains 45 states.

A. PROPERTIES CHECKING

Let us now formulate two different queries over the model. First, we would like to check if it is possible to get the results from the workflow's execution without specifying an email address. This query corresponds to the following CTL predicate: $EG(AND(NOT(RDFGmail), EF(RDFGresult)))$, where RDFGmail and RDFGresult are two RDF graphs that specify if there exists a path in which for all its states there is no information about the email while the result is calculated and propagated up to the end. We want to check if this formula fails, so the answer to the question will positive. The model does not verify the formula, so the answer to the query is TRUE. As a result, the COMBAS tool produced the following output:

```
ERROR [main] (CombasApp.java:241) -
  Model doesn't satisfy the formula.
INFO [main] (CombasApp.java:244) -
  Output files generated.
INFO [main] (CombasApp.java:248) -
  Checked in 142 millis
INFO [main] (CombasApp.java:249) -
  Formula: formula_qdlgfxyo3yn2
INFO [main] (CombasApp.java:250) -
  Model: netId2p7pw09m4j1x_RG
```

As a second example, let us now check if it is possible to get the results from the protein analysis in GFF format but not as a formatted text. This query can be expressed with the following formula: $EG((RDFGPgff) \rightarrow AF(RDFGPtxt))$, stating that there exist a path in which every state verifies one or both of these conditions: i) RDFGPgff is not verified, which implies that the result has not been calculated in GFF format, so the formula is true at these states, or ii) RDFGPgff and AF(RDFGPtxt) are both verified, which means that from that point, for every path there exist a future state where the solution is formatted as text.

Therefore, the formula will be true if: i) the solution is not formatted as GFF, or ii) the solution is calculated in GFF and text formats. The way the net in Figure 10 has been modeled produces the formula to be satisfied, since the result is first formatted as text and then transformed to GFF. The model satisfies the formula, so the answer to the query is false. This was checked with the COMBAS tool:

```
INFO [main] (CombasApp.java:239) -
  Model satisfies the formula!
INFO [main] (CombasApp.java:244) -
  Output files generated.
INFO [main] (CombasApp.java:248) -
  Checked in 1186 millis
INFO [main] (CombasApp.java:249) -
  Formula: formula_e2bi82zmwasx
INFO [main] (CombasApp.java:250) -
  Model: netId2p7pw09m4j1x_RG
```

Another different inconsistences of the input data could be detected using the approach presented in this paper. For example, if the output of a service returns the information about the structure of some proteins in angstroms being this result used in a later step as the input of a service requiring the used of millimeters, the analysis could detect this situation, avoiding the problem to be detected at execution time, after a few hours of computations! The proposed approach also allows detecting problems without knowing specific data, detecting two properties of a concept to be different. Let us, for instance, suppose a service needs a Unit Cell containing the same parameters for Alpha and Beta. This can be represented in our model with a transition whose input arcs contain the RDF graph patterns `:_ProteinComplex po:Alpha ?Alpha` and `:_ProteinComplex po:Beta ?Beta` and the guard `?Alpha≠?Beta`. If the input data of the transition contain the triples `:_ProteinComplex po:Alpha :_Alpha` and `:_ProteinComplex po:Beta :_Alpha` it can be ensured without execution that the experiment will fail (because the guard will be evaluated to false).

VII. CONCLUSIONS

Petri nets and model checking techniques are widely used in different application domains. This work has focused on their application to the area of scientific workflow analysis. The use and adaptation of traditional model checking techniques to this area not only allows checking interesting properties and behaviors of a workflow prior to its execution, but also the safe modeling and implementation of reliable systems without any additional cost in terms of computation or physical resources.

Adding semantic aspects to workflow models allows a higher flexibility for analysis and improves resource usage when dealing with complex problems. However, this requires some specific considerations at both the modeling and analysis stages. This has been the main focus of this paper, where the U-RDF-PN has been presented as a high level formalism for the modeling of scientific workflows with well defined semantics, which allows the set of system states and state transitions to be generated and used as the inputs of the model checker. In order to prove the feasibility of the proposed approach, a fully-functional environment for model checking has been implemented.

The suitability of the approach presented in this work has been demonstrated by means of its application to two different cases. On the one hand, the First Provenance Challenge (FPC) workflow has been used to introduce the main concepts related to semantics and model checking.

On the other hand, the InterScan protein analysis workflow has been presented to depict an example in the field of biocomputing engineering. Both scenarios are educative enough as to allow presenting the main concepts and techniques behind our proposal as well as exemplifying its application.

However, the proposed method has some important limitations. For a given task specification there are different types of postconditions that could be defined. Many of them could be evaluated without executing the task invocation, and there are sets of postconditions that could only be evaluated by means of the task execution. These cases cannot be considered if we want the U-RDF-PN model as a mean to foresee the system behavior. For instance, in the First Provenance Challenge Workflow, if a task specification states that the output format of an image is JPEG, or it is the same of the input image, and the initial marking of the net establishes the initial parameter that will feed the workflow, it would be possible to evaluate the transition enabling and the algorithm will construct the reachability graph allowing the workflow analysis. However, there are sets of postconditions that require the task to be executed in order to get them. For instance, when some output parameter depends on data received at run-time or refers to the result of the task invocation (for instance, *the first element of the resulting vector is the negative value of the first input parameter*).

On the other hand, another limitation is that our technique is not able to deal with preconditions requiring results that would only be known after the real task execution. For instance, let us suppose that in the EBIs WSInter-ProScan service example the runInterProScan service has a precondition in which is established that this service will be executed only if the parameters describing crystallographic data verify some property, for example `po:Alpha` and `po:Beta` are over 50 angstroms. In such case, our model is not able to answer if this precondition is true or false because this data will depend of the final experiment, and therefore it is not able to build the reachability graph.

A solution to tackle these issues is being addressed: the presented formalism and the analysis techniques are being extended in order to deal with parametric data in preconditions allowing this way to analyze a wider number of scientific workflows [37].

Our ongoing work is also focusing on the efficiency of the prototype, which is being improved to deal with more complex systems, as experimental results demonstrated the existence of a bottleneck in the computation of the reachability graph. We have already developed a parallel version of the tool that is executed in cluster and cloud environments, greatly decreasing computation time [44]. Finally, the COMBAS framework allows the use of different RDF storages, so we are carrying out a study to analyze and improve the efficiency of the overall system as each RDF solution exposes different costs depending on the inference engine.

REFERENCES

[1] C. Berkley, S. Bowers, M. B. Jones, B. Ludäscher, M. Schildhauer, and J. Tao, "Incorporating semantics in scientific workflow authoring," in *Proc. 17th Int. Conf. Sci. Stat. Database Manage. (SSDBM)*. Berkeley, CA, USA: Lawrence Berkeley Nat. Lab, 2005, pp. 75–78.

[2] M. C. Cavalcanti *et al.*, "Managing structural genomic workflows using Web services," *Data Knowl. Eng.*, vol. 53, no. 1, pp. 45–74, 2005.

[3] A. Preece *et al.*, "Managing information quality in e-science using semantic Web technology," in *Proc. 3rd Eur. Semantic Web Conf. (ESWC)*, 2006, pp. 472–486.

[4] Z. Lacroix, C. R. L. Legendre, and S. Tuzmen, "Reasoning on scientific workflows," in *Proc. IEEE Congr. Services (SERVICES)*, Jul. 2009, pp. 306–313.

[5] C. A. Goble *et al.*, "myExperiment: A repository and social network for the sharing of bioinformatics workflows," *Nucleic Acids Res.*, vol. 38, pp. W677–W682, Jul. 2010.

[6] K. Derouiche and D. A. Nicole, "Semantically resolving type mismatches in scientific workflows," in *Proc. Move Meaningful Internet Syst.*, Vilamoura, Portugal, Springer-Verlag, 2007, pp. 125–135.

[7] J. Kim, Y. Gil, and V. Ratnakar, "Semantic metadata generation for large scientific workflows," in *Proc. 5th Int. Semantic Web Conf. (ISWC)*, 2006, pp. 357–370.

[8] S. Munroe, S. Miles, L. Moreau, and J. Vazquez-Salceda, "PrIMe: A software engineering methodology for developing provenance-aware applications," in *Proc. 6th Int. Workshop Softw. Eng. Middleware (SEM)*, 2006, pp. 39–46.

[9] Y. L. Simmhan, B. Plale, and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Sep. 2006, pp. 427–436.

[10] S. M. S. da Cruz, M. Luiza, M. Campos, and M. Mattoso, "Towards a taxonomy of provenance in scientific workflow management systems," in *Proc. Congr. Services (SERVICES)*, Jul. 2009, pp. 259–266.

[11] T. Ellqvist, D. Koop, J. Freire, C. Silva, and L. Strömbäck, "Using mediation to achieve provenance interoperability," *Proc. IEEE Congr. Services (SERVICES)*, Jul. 2009, pp. 291–298.

[12] V. Curcin, M. Ghanem, and Y. Guo, "The design and implementation of a workflow analysis tool," *Philos. Trans. Roy. Soc. A, Math. Phys. Eng. Sci.*, vol. 368, no. 1926, pp. 4193–4208, 2010.

[13] Y. He, G. Liu, D. Xiang, J. Sun, C. Yan, and C. Jiang, "Verifying the correctness of workflow systems based on workflow net with data constraints," *IEEE Access*, vol. 6, pp. 11412–11423, 2018.

[14] M. D. Wilkinson, B. Vandervalk, and L. McCarthy, "The semantic automated discovery and integration (SADI) Web service design-pattern, API and reference implementation," *J. Biomed. Semantics*, vol. 2, no. 8, pp. 1–24, 2011.

[15] D. Newman, S. Bechhofer, and D. De Roure, "myExperiment: An ontology for e-research," *Semantic Web Appl. Sci. Discourse*, Aug. 2009.

[16] J.-H. Pfeiffer, W. R. Rossak, and A. Speck, "Applying model checking to workflow verification," in *Proc. 11th IEEE Int. Conf. Workshop Eng. Comput.-Based Syst. (ECBS)*, May 2004, pp. 144–151.

[17] F. L. Tiplea, D. C. Marinescu, and C. Lin, "Model checking and abstraction for workflow net verification," in *Proc. 1st Int. Workshop Petri nets Coordination (PNC)*, 2004, pp. 131–145.

[18] H. Huang and R. A. Mason, "Model checking technologies for Web services," in *Proc. 4th IEEE Workshop Softw. Technol. Future Embedded Ubiquitous Syst. (SEUS-WCCIA)*, Apr. 2006, p. 6.

[19] R. Dong, Z. Wei, and X. Luo, "Model checking behavioral specification of BPEL Web services," in *Proc. World Congr. Eng.*, vol. 1, 2008, pp. 383–399.

[20] S. Patig and M. Stolz, "A pattern-based approach for the verification of business process descriptions," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 58–87, 2013.

[21] M. Haydar, A. Petrenko, S. Boroday, and H. Sahraoui, "A formal approach for run-time verification of Web applications using scope-extended LTL," *Inf. Softw. Technol.*, vol. 55, no. 12, pp. 2191–2208, 2013.

[22] G. Liu, W. Reisig, C. Jiang, and M. Zhou, "A branching-process-based method to check soundness of workflow systems," *IEEE Access*, vol. 4, pp. 4104–4118, 2016.

[23] M. J. Ibanez, J. Fabra, P. Alvarez, and J. Ezpeleta, "Model checking analysis of semantically annotated business processes," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 4, pp. 854–867, Jul. 2012.

[24] L. Moreau *et al.*, "Special issue: The first provenance challenge," *Concurrency Comput., Pract. Exper.*, vol. 20, no. 5, pp. 409–418, 2008, doi: 10.1002/cpe.v20:5.

[25] I. Paik, W. Chen, and M. N. Huhns, "A scalable architecture for automatic service composition," *IEEE Trans. Services Comput.*, vol. 7, no. 1, pp. 82–95, Jan./Mar. 2014.
[26] D. Brickley and R. V. Guha. (Feb. 2014). *RDF Vocabulary Description Language W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/rdf11-primer/
[27] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. (Feb. 2014). *RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/rdf-schema/
[28] P. Hayes. (2012). *Rudolph OWL Web Ontology Language Primer, W3C Recommendation, World Wide Web Consortium*. [Online]. Available: https://www.w3.org/OWL/
[29] P. Hayes, "RDF semantics," World Wide Web Consortium, Tech. Rep. W3C, Feb. 2014. [Online]. Available: https://www.w3.org/TR/rdf11-mt/
[30] E. P. Hommeaux and A. Seaborne. (Jan. 2008). *SPARQL Query Language for RDF W3C Candidate Recommendation World Wide Web Consortium*. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/
[31] S. Schenk, P. Gearon, and A. Passant. *SPARQL 1.1. Technical Report, W3C 2013*. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/
[32] W. M. P. van der Aalst, "Three good reasons for using a Petri-Net-based workflow management system," in *Proc. Int. Work. Conf. Inf. Process Integr. Enterprises (IPIC)*, 1996, pp. 161–182.
[33] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits, Syst. Comput.*, vol. 8, no. 1, pp. 21–66, 1998.
[34] K. M. van Hee, N. Sidorova, and J. M. van der Werf, "Business process modeling using Petri nets," in *Transactions on Petri nets and Other Models of Concurrency VII*, (Lecture Notes in Computer Science), vol. 7480. Berlin, Germany: Springer, 2013, pp. 116–161.
[35] T. Gubala, D. Herezlak, M. Bubak, and M. Malawski, "Semantic composition of scientific workflows based on the Petri nets formalism," in *Proc. 2nd IEEE Int. Conf. e-Sci. Grid Comput. (E-SCIENCE)*, Dec. 2006, pp. 1–12.
[36] Z. Guan *et al.*, "Grid-flow: A grid-enabled scientific workflow system with a Petri-Net-based interface," *Concurrency Comput., Pract. Exper.*, vol. 18, pp. 1115–1140, Dec. 2005.
[37] M. J. Ibáñez, P. Álvarez, and J. Ezpeleta, "Analyzing behavioral properties of semantic business processes with parametric data," *Concurrency Comput., Pract. Exper.*, vol. 23, pp. 525–555, Apr. 2011.
[38] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[39] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent system using temporal logic specifications: A practical approach," in *Proc. 10th ACM SIGACT-SIGPLAN Symp. Princ. Program. Lang. (POPL)*, 1983, pp. 117–126.
[40] E. González-López de Murillas, "Temporal and modal logic," in *Handbook of Theoretical Computer Science* (Formal Models and Semantics). Cambridge, MA, USA: MIT Press, 1990, pp. 995–1072.
[41] E. González-López de Murillas, J. Fabra, P. Álvarez, and J. Ezpeleta, "COMBAS: A semantic-based model checking framework," in *Proc. 6th Int. Conf. Adv. Eng. Comput. Appl. Sci. (ADVCOMP)*, 2012, pp. 46–52.
[42] J. Billington *et al.*, "The Petri net markup language: Concepts, technology, and tools," in *Proc. 24th Int. Conf. Appl. Theory Petri Nets (ICATPN)*, 2003, pp. 483–505.
[43] A. Rutle, F. Rabbi, W. MacCaull, and Y. Lamo, "A user-friendly tool for model checking healthcare workflows," *Procedia Comput. Sci.*, vol. 47, no. 5, May 2017.
[44] E. González-López de Murillas, J. Fabra, P. Álvarez, and J. Ezpeleta, "Parallel computation of the reachability graph of Petri net models with semantic information," *J. Softw., Pract. Exper.*, vol. 47, no. 5, pp. 647–668, 2017.
[45] E. M. Zdobnov and R. Apweiler, "InterProScan—An integration platform for the signature-recognition methods in InterPro," *Bioinformatics*, vol. 17, no. 9, pp. 847–848, 2001.
[46] D. A. Natale *et al.*, "The protein ontology: A structured representation of protein forms and complexes," *Nucleic Acids Res*, vol. 39, pp. D539–D545, Jan. 2011.
[47] C. J. Bult *et al.*, "The representation of protein complexes in the protein ontology (PRO)," *Bioinformatics*, vol. 12, no. 1, p. 371, 2011.

JAVIER FABRA received the Ph.D. degree in computer science from the University of Zaragoza, Spain, in 2010. He has been an Associate Professor with the Department of Computer Science and Systems Engineering, University of Zaragoza, since 2008. His main research areas focus on service-oriented computing and cloud architectures, semantic and scientific computing, and interoperability issues in cluster, grid, and cloud scenarios.

MARÍA JOSÉ IBÁÑEZ received the M.S. degree in mathematics from the University of La Rioja, Spain, in 2006, and the Ph.D. degree in computer-science engineering from the University of Zaragoza, Spain, in 2011. Her research interests include service oriented computing, machine learning, and semantic Web.

PEDRO ÁLVAREZ received the Ph.D. degree in computer science engineering from the University of Zaragoza, Zaragoza, Spain, in 2004. He has been a Lecture Professor with the University of Zaragoza since 2000. His current research interests focus on two main aspects—the integration problems of network-based system and the use of novel techniques and methodologies for solving them and the application of formal analysis techniques to mine event logs and databases.

JOAQUÍN EZPELETA received the M.S. degree in mathematics and the Ph.D. degree in computer science from the University of Zaragoza, Spain. He is currently a Professor of the Department of Computer Science and Systems Engineering, University of Zaragoza, where he conducts lectures on formal methods for sequential and concurrent programming and service-oriented architectures. His research focuses on the problem of modeling, analysis, and control synthesis for concurrent systems, the application of formal techniques to help in the development of correct distributed systems based on Internet and cloud technologies, and the parallel processing of data and computing intensive computing problems.