International Journal of Data Science and Analytics (2021) 11:361–376
https://doi.org/10.1007/s41060-020-00237-x

REGULAR PAPER

A workflow language for research e-infrastructures

Leonardo Candela[1] · Valerio Grossi[2] · Paolo Manghi[1] · Roberto Trasarti[1]

Received: 17 July 2019 / Accepted: 14 October 2020 / Published online: 11 February 2021

Abstract
Research e-infrastructures are "systems of systems," patchworks of resources such as tools and services, which change over time to address the evolving needs of the scientific process. In such environments, researchers carry out their scientific process in terms of sequences of actions that mainly include invocation of web services, user interaction with web applications, user download and use of shared software libraries/tools. The resulting workflows are intended to generate new research products (articles, datasets, methods, etc.) out of existing ones. Sharing a digital and executable representation of such workflows with other scientists would enforce Open Science publishing principles of "reproducibility of science" and "transparent assessment of science." This work presents HyWare, a language and execution platform capable of representing scientific processes in highly heterogeneous research e-infrastructures in terms of so-called hybrid workflows. Hybrid workflows can express sequences of "manually executable actions," i.e., formal descriptions guiding users to repeat a reasoning, protocol or manual procedure, and "machine-executable actions," i.e., encoding of the automated execution of one (or more) web services. An HyWare execution platform enables scientists to (*i*) create and share workflows out of a given action set (as defined by the users to match e-infrastructure needs) and (*ii*) execute hybrid workflows making sure input/output of the actions flow properly across manual and automated actions. The HyWare language and platform can be implemented as an extension of well-known workflow languages and platforms.

Keywords Workflow languages · Scientific workflows · Open Science · HyWare

1 Introduction

Over the past decade Europe has developed world-leading expertise in building and operating e-Infrastructures [29,34]. They are large scale, federated and distributed research environments in which researchers have shared access to unique scientific facilities (including data, instruments, computing and communications), regardless of their type and location in the world. They are meant to support unprecedented scales of international collaboration in science, both within and across disciplines. Their aim is to realize a common environment where scientists can create, validate, assess, compare and share their digital results of science, such as research data, intended as scientific data produced by a scientific effort, and research methods, intended as digital computation-oriented elements resulting from their research; research methods are discipline-specific, as well as research data, but examples include software, services, tools, workflows, scripts, algorithms and protocols.

The digitalization of the scientific process has raised unprecedented opportunities and challenges in the way science can be performed but also shared, evaluated and reused. In the last decade, all stakeholders of the research life cycle (e.g., researchers, organizations, funders) have highlighted and endorsed the importance of applying Open Science publishing principles [4]. According to such principles, researchers should "publish" their scientific results in order to enable reuse (reducing the cost of science), reproducibility of science and transparent evaluation of science. According to such vision, the scientific article is only one of the possible publishable products, certainly required but insufficient

✉ Leonardo Candela
leonardo.candela@isti.cnr.it

Valerio Grossi
vgrossi@di.unipi.it

Paolo Manghi
paolo.manghi@isti.cnr.it

Roberto Trasarti
roberto.trasarti@isti.cnr.it

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" Consiglio Nazionale delle Ricerche, Pisa, Italy

[2] Dipartimento di Informatica, Università di Pisa, Pisa, Italy

at supporting Open Science principles. The Open Science movement encourages researchers to share (as openly as possible) (*i*) the digital products such as *research data* and *research methods* valuable to their research (e.g., input and output data, a text-mining algorithm), (*ii*) the e-infrastructure *tools* they used to implement their research (e.g., a facility service to run methods over data) and possibly (*iii*) the *research workflows*, intended as sequences of actions they performed to reach their results.

Research workflows are the representation of the scientific process, and the steps the researchers had to perform using the e-infrastructure tools to run an experiment (e.g., "I used tool X to test my method Y over research data Z and obtained research data W"). In most scenarios workflows are described in the scientific article, which, together with the availability of all digital products and tools, maximizes the possibility of reproducing an experiment and objectively evaluate the quality of the underlying research. The optimal scenario, however, is one where the scientific process can be encoded in terms of digital, fully automated and executable workflows: digital research products, which provided together with the related products, can be shared to facilitate the process of reproducibility of science [21].

Research infrastructures today are often far from being well-designed and consistent environments embracing Open Science needs. They are rather "systems of systems," patchworks of tools that process or generate research products, often equipped with "catalogues" where researchers (and services) can register both research tools and products to enable their sharing, discovery and reuse. In such contexts, the heterogeneity of tools is often a necessity as it is not possible to assume that scientists will produce e-infrastructure tools adhering to a rigorous workflow language-like approach (e.g., Apache Taverna [25], Galaxy [18], KNIME [6]). For example, actions of a workflow may be: first invoke a web service A over data d_1 to produce the result d_2; then download a tool B for the execution of R-scripts and execute the script r over the data d_2 to obtain d_3; finally, take the Taverna workflow t and execute it on myExperiment.org over d_3. Indeed, while subsystems of e-infrastructures may support workflow languages and engines, this way of thinking can be hardly imposed to the research e-infrastructure as a whole, namely cross-platform, cross-nation, cross-laboratory, cross-funders, etc. Workflows, in such context, remain confined into the article narration as a natural language description of how tools must be combined to obtain the expected results. Although the availability of other products represents an important step forward toward Open Science, reproducibility is severely compromised by the absence of digital and executable experiments [9,31].

Paper contribution. This work presents HyWare (HYbrid Workflow lAnguage for Research E-infrastructures) [8], a language and an execution platform for representing scientific process in highly heterogeneous e-infrastructures in terms of so-called *hybrid workflows*, which can express sequences of *manually executable actions* and *machine-executable actions*. In other words, HyWare lays in between business process modeling languages [5,33], which offer a formal and high-level description of a reasoning, protocol or procedure, and workflow execution languages (e.g., BPEL [33]), which enable the fully automated execution of sequences of computational steps via dedicated engines.

HyWare supports a framework where research actions templates (identifying classes of actions) can be customized according to the specifics of the underlying e-infrastructure and then be combined by researchers into workflows. An HyWare platform, i.e., a system build to support management of HyWare workflows, offers user interfaces to support the scientists at constructing a workflow out of the available action classes and make it available for others to discovery. A second scientist may later discover the workflow and, via the same HyWare-based tool, be guided through its execution on a step-by-step basis. Such tools display to the researchers which steps they should manually execute to repeat the experiments but, when a sequence of steps is based on components of executable workflows, execute automatically the relative actions.

Outline. The rest of the paper is organized as follows. Section 2 discusses related works. Section 3 provides motivations for the definition of a "hybrid" workflow language such as HyWare by introducing an example and some terminology. Section 4 formally defines terminology, elements and semantics of the HyWare language, together with a sketch of a possible implementation. Section 5 describes the HyWare platform designed to support the execution of HyWare workflows by presenting the main components and discussing a KNIME-based implementation. Section 6 exemplifies the usage of HyWare in the real-case scenario of the SoBigData.eu Research Infrastructure [15]. Finally, Sect. 7 concludes the paper by reporting on future works.

2 Related work

Liew et al. [22] have recently analyzed selected workflow management systems (WMSs) that are widely used by the scientific community, namely: Airavata [24], Kepler [20], KNIME [6], Meandre [23], Pegasus [11], Taverna [35] and Swift [36]. Such systems have been analyzed with respect to a framework aiming at capturing the major facets characterizing WMSs: (*a*) processing elements, i.e., the building blocks of workflows envisaged to be either web services or executable programs; (*b*) coordination method, i.e., the mechanism controlling the execution of the workflow elements envisaged to be either orchestration or choreography; (*c*) workflow representation, i.e., the specification of a work-

flow that can meet two goals human representation and/or computer communication; (*d*) data processing model, i.e., the mechanism through which the processing elements process the data that can be bulk data or stream data; (*e*) optimization stage, i.e., when optimization of the workflow (if any) is expected to take place that can either be build time or runtime (e.g., data workflow processing optimization [19].

In the era of Open Science, where aspects such as reproducibility and transparency of science and FAIRness of research data (Findable, Accessible, Interoperable, Reusable research data)[1] are becoming central to the whole research life cycle, workflow languages play a special role. Workflows can inherently contribute to the implementation of FAIR data principles by accurately collecting, processing and managing data and metadata on behalf of the researchers, while tracking provenance according to standards [17]. Moreover, workflows are digital objects in their own right, and they can be published, discovered, shared and cited for reproducibility and for scientific attribution of science like research articles, research data and research software. Known approaches include (*i*) *workflows as digital artifacts*: workflow files are published in a repository with bibliographic metadata (e.g., Zenodo[2], MyExperiment.org [27]) and can be possibly related to their inputs and outputs [14,30] ; (*ii*) *workflow-as-a-Service*: workflows are shared via a platform/science gateway that enables their discovery and execution [10,12].

However, the aforementioned approaches are defined based on the assumption that workflows are composed of machine-executable actions, i.e., performed by agents that can be programmatically invoked. They do not address the needs, motivated by several scientific contexts, e.g., Big Data and Social Mining [16], Biodiversity and Cheminformatics domains [13,28], of defining workflows that include "manual actions" (cf. Sec. 3), e.g., data manipulation and adaptation using editors or shell commands. Attempts in this direction exist but embrace a fully manually oriented approach, e.g., protocols.io [32], enabling the digital representation, publishing and sharing of digital fully manual workflows.

The main contribution of this paper is the HyWare workflow language and execution platform, whose intuition was earlier presented in [8]. HyWare is uniquely designed to enable the description of "hybrid" workflows, obtained as sequences of machine-executable and manually executable actions. As such, the language can serve the mission of Open Science by addressing reproducibility of digital science beyond traditional approaches, in contexts where workflow actions are not entirely performed by machines. In this paper, the language is formalized and flanked by a proposed implementation of its orchestration platform (cf. Sec. 5.1.3), realized as an extension of the KNIME language [6]. This makes HyWare interoperable with other known workflow language platforms [14,26] like CWL [1]. Moreover, the HyWare orchestration platform allows users to create, execute, monitor and publish workflows via a GUI (cf. Sec. 5.1.1) to reduce the learning curve and support the execution of sequences of manual and machine-executable actions.

3 Rationale

One of the aims of a research e-infrastructure (e-infra) is to guarantee an integrated framework in order to provide the researchers with an homogeneous and expressive way for representing experiments, performing and sharing them among the community. Consider the following scenario: a scientist working with the e-infra is using the tools it provides to run the experiments, which consist in reusing and generating research data and methods by means of sequences of actions of the following kinds:

- *Local execution of software to be first downloaded and installed*: the execution of the action requires the user to download and execute a software on its own premises;
- *Call to web-accessible services (SOAP/REST)*: the execution of the action requires a call to a remote service;
- *Web-accessible applications (tools accessible via user interfaces from the web)*: the execution of the action requires accessing a web user interface to use a given functionality;
- *Manual operations*: the execution of the action consists in performing an operation that does not involve e-infrastructure resources of the kinds above, but it is mandatory to complete the experiment (e.g., "make an output file available at given URL," "process an input CSV file to remove column X and return the resulting CSV file").

Once the experiment is concluded, the scientist has identified the actions he/she needs to perform to reproduce it and she is now willing to materialize the relative workflow in order to share it with others. This will allow other researchers to be convinced of the quality and value of her scientific process, by possibly repeating and reproducing the experiment for validation purposes or reusing its parts. To this aim, the scientists need a workflow language capable of describing and combining machine-executable actions and human-executable actions in such a way the workflow can be re-executed in a trustable and objective way, guaranteeing evaluation and reproducibility of science. This language has to orchestrate actions resulting from tools and best practices made available to the e-infra and interpreted by different scientists, with no common agreements and policies on the way

[1] FAIR principles of research data, https://www.force11.org/group/fairgroup/fairprinciples

[2] https://zenodo.org/

the flow between such actions has to be implemented, e.g., how input and output data flows from one action to another.

HyWare (HYbrid Workflow lAnguage for Research E-infrastructures) [8] is a workflow modeling language designed to meet such scenarios. Its execution engine performs the orchestration of the workflow by delegating the execution of some of the actions to the e-infra, of other actions to the scientists, and ensuring input/output parameters of such actions are properly channeled. HyWare takes inspiration from:

- *Orchestration language*: HyWare graphically describes a workflow as a directed graph of nodes whose input and output parameters are interconnected and validated according to a degree of compatibility, and whose edges imply a temporal ordering of execution; workflows are executed by a local engine, which executes their business logic and lets the data flow across them, to produce a final output;
- *The protocols.io approach* [32]: HyWare actions that require human interaction are described according to given templates, so that other scientists can interpret them and execute them; such actions are executed by the HyWare engine, which prompts the scientist via GUI with the instructions to perform the action; also human actions obey to minimal specification requirements, which allow input and output of the actions to flow from machine executable to human executable nodes.

HyWare hybrid workflows are built out of two main classes of action nodes: (*i*) the actions that require human interaction, and (*ii*) the actions that are performed by the e-infrastructure. Unlike other languages, however, the workflow makes sure that research data or research methods flow through both classes of nodes correctly, by adhering to static compatibility patterns. As in the case of protocols.io the language does not enforce specific kinds of actions, but rather gives the possibility to instantiate e-infra specific classes of human actions (e.g., download and execution of software) or machine actions (e.g., web service call) based on a meta-structure for actions:

Human actions. Human actions are characterized by a description, expressed by the respective properties, which should be detailed enough to guide a scientist through the execution of the action. As a general design pattern, human actions should reflect the flavor of machine actions, in the sense their execution should ideally return output parameters after processing input parameters. For example, the operation of downloading software and installing should not be regarded as a "proper" HyWare human action. The action of downloading a software, installing it, executing it over an input data file and return an output data file, can be instead considered a valuable HyWare action.

Machine actions. Machine actions are characterized by a description, expressed by the respective properties, but are also associated with a specific implementation of an *action actuator*. An actuator is a method whose implementation wraps the business logic of the action (i.e., each action class has its own actuator implementation): it is invoked with the input parameters of the action to execute the business logic of the action (e.g., invoking a third-party service or local code) and returns the results to return the output parameters of the action. For example, a machine action class corresponding to the execution of a REST service X implements the actuator as a wrapper that properly maps the input parameters, performs the call to the service and maps the output of the service as an output of the actuator.

Hybrid workflows. A hybrid workflow is a composition of human and/or machine actions into a Directed Acyclic Graph (DAG). Time edges between actions express the chronological ordering of the actions, hence, before an action can begin all actions relative to incoming time edges must be terminated. As in the case of service workflow languages, between two nodes connected by a time edge other input–output edges exist, indicating the associations between the output parameters and the input parameters of the two actions.

In the following we outline a simple example of a hybrid workflow using different processing tools provided by an e-infrastructure.

Example 1 (Proactive carpooling application)

This example shows a proactive carpooling application, developed by means of the tools for analysis of city mobility (or city of citizens) listed in Table 1. For example, the Trajectory Builder operator is a *web service* hosted by the e-infrastructure, which can be executed on demand in the cloud; the Urban Mobility Atlas is available both as a *web service*, for programmatic access, or as a *web application*, for end-user access. Anyway most of the services make their *software* freely download-able for local execution. Researchers may combine and configure these tools to build their own analytic processes. This mapping between tools of the e-infrastructure and the relative actions allows the user to use HyWare inside the VREs to represent the analytic processes they perform in terms of hybrid workflows, i.e., DAGs of actions. Such workflows are themselves e-infra products, hence shareable and reproducible, by other scientists, as well as subject to comments and discussion.

In this particular example we present a workflow for the analysis of vehicular GPS traces in order to build a proactive carpooling application. The workflow performs an analytic process that extracts the systematic movements of the people and computes the possible matches between "compatible users" offering them an automatic list of recommendations. In Fig. 1, high-level representation of the workflow is depicted. The researcher has chosen the tools

Table 1 Example of City of Citizens operators

Action	Type of invocation
Urban Mobility Atlas	Web page, Web service
Trajectory Builder	Service hosted, Download
Borders	Web service
Sociometer	Download
Trip Builder	Web page
Carpooling	Download
MyWay	Download
Privacy Risk	Download
O/D Matrix	Web service
Mobility Profiles	Download
Exploration of Time	Download
Statistical Validation	Service hosted, Download
Twitter Scraper	Web service
IGD Graph Visualizer	Web page

needed for his analytic process. Building it, he discovered that he needs some additional steps of *data transformation* where the results of some tools must be adapted to be processed by the next one. Moreover, as already said, different tools are invoked in a different way.

The process starts with the *Trajectory Builder* action which require a reference to a table in the platform database containing the spatio-temporal observations of the user in the form: userid, the longitude, latitude and the timestamps. The result is a sequence of trajectories followed by the vehicles separated in the case a stop is identified.

The resulting data must be processed by the user exporting the data from the database into a CSV file and then *manually* transformed to fit the format required by the next tool.

Mobility Profile is a tool available for local execution; therefore, the user must first download it, then install it and finally execute it on the data he prepared in the previous step. All the trajectories of a single vehicle are grouped in order to extract a concise representation of its systematic movements. The obtained result is in the right format, so no transformations are needed.

Carpooling is available for local execution as well and the user needs to follow a similar procedure to the previous step. The result is a list of vehicle pairs representing the vehicles that may give a ride to the other. This procedure is done over the systematic movements; therefore, the match defines the vehicles which are compatible in the everyday activities. The results produced by the carpooling software have to be transformed in order to be visualized by the IGD Graph Visualizer. In practice from a specific format the user must transform the data in a JSON file. The user then may visualize the graph obtained and apply visual analytic tools to extract the final list of matches between passengers and drivers.

This example clearly highlights in a real-case scenario the need for hybrid workflows to encode a repeatable scientific processes. In the following sections we provide a formalization of HyWare actions and workflows and describe how an HyWare platform installation can be used as an add-on to an existing research e-infrastructure, highlighting the integration requirements and required efforts, so as to support such kind of workflows.

4 The workflow language

In this section we formally describe an *Action class*, an *action instance*, how the latter can be combined into a *well-formed workflow*, and how such workflows are executed by an *engine* via a shared *memory*. An engine underpins the execution of workflows by ensuring input and output parameters flow consistently across the relative actions to reach completion.

Definition 1 (Action class) An *Action class A* is a representation of one operation, i.e., a manual or a machine action, that researchers typically need to perform to implement their scientific process. Formally, it is defined in terms of inputs and outputs, processing function and configuration parameters, $A =< in, out, f, cp >$:

$$in = \{p_1^i, \ldots, p_n^i\}, out = \{p_1^o, \ldots, p_k^o\}, cp = \{l^1, \ldots, l^r\}$$

where p_i^* is a port defined by the triple $< type, loc, opt >$, representing, respectively: the *type* of data produced (output) or required (input), the *location* of the data (e.g., local file system, remote file system, database, etc.), and the *opt* flag defining if the port is *optional* or not. Ports are the channels through which actions acquire inputs from preceding actions and deliver outputs to subsequent actions. Configuration parameters are values to be manually provided by the scientists, required by the action to perform f. De facto, an Action class represents the generic pattern $out = f(in, cp)$, where f is the business logic of the human or machine action.

A port type is a unique label representing a domain of known values in the e-infrastructure. As it is expected from programming language types, values of such domains conform to given meaning and/or structure. On the other hand, ports are not always expected to carry such values, like variables in programming language do. Depending on the port location loc, ports may contain values of the following kinds:

- *URLs to remote files* ($loc =\uparrow$): the file is expected to contain a value of the given type and it is the consuming action that needs to fetch such file;
- *URIs to local files* ($loc =\downarrow$): as above, the file is expected to contain a value of the given type and it is the consuming action that needs to fetch such file;

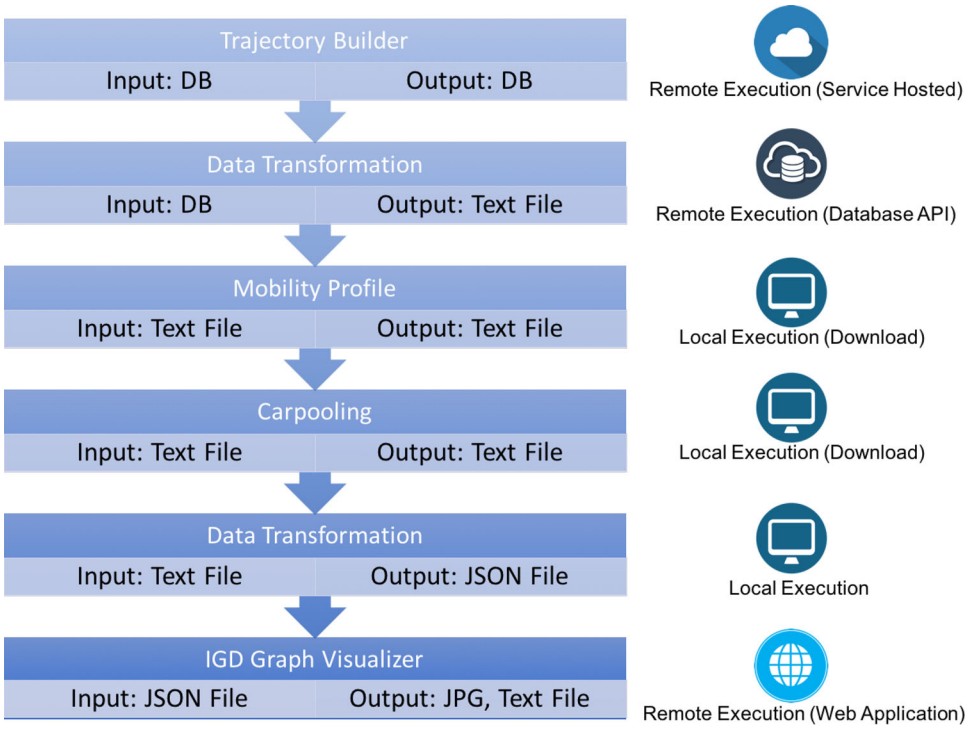

Fig. 1 Proactive carpooling application process

– *an actual value* ($loc = -$) represented as a string: the value needs to be cast by the consuming action to conform to the given type, for example a string, an integer, a Boolean or a custom type.

If an action class A has an output port p^o, associating it with location kinds $\uparrow$ or $\downarrow$ determines a specific behavior of the function $A.f$. For example if $p^o.loc = \uparrow$ then f is expected to upload a file of type $p^o.type$ on the Web at a URL to be passed via p^o. An actual value location $-$ implies that the value passed over the port has a specific interpretation (and consumption modes) shared across a set of Action classes and identified by the type. For example, a set of action classes may define functions f's operating over a shared database whose configuration and access are embedded in the business logic of the actions; actions may define a custom-type $remoteDB$ whose actual values are strings representing valid SQL queries. In another scenario, the same set of actions may instead be able to access any SQL database accessible online; hence, the type $remoteDB$ would be associated with values as queries expressed by URLs that include database connection details.

Definition 2 (Compatible ports) An input port p^i and output port p^o are *compatible* ($p^o \rhd p^i$) iff

$p^o.type \leq p^i.type \wedge p^o.loc \sqsubseteq p^i.loc$

where

$p^i.type = p^o.type \implies p^i.type \leq p^o.type$ and $p^i.loc = p^o.loc \implies p^i.loc \sqsubseteq p^o.loc$

but other custom compatibility statements can be added by HyWare users to match specific e-infrastructure and actions needs. This means that types can be casted to application-specific compatibility rules based on $\leq$.

As such, an action class is simply a "template" of an operation, whose actual execution is instead represented by an *action instance*.

Definition 3 (Action instance) An *action instance* a, or simply an *action*, is one specific instance of an Action class A. The same action class may have multiple instances, as many as needed by a scientists to perform a scientific process.

Action instances make sense as part of a so-called *workflow*, involving other actions of the same of different action classes. An action instance has a Boolean flag $executed$ that represents its state of execution.

Definition 4 (Workflow) Given a set of Action classes $A = \{A_1, \ldots, A_m\}$, a *workflow* w can be defined as a directed acyclic graph $w = (N, E)$ where nodes $N = \{a_1, \ldots, a_n\}$ are action instances of the classes in A and edges $E = \{e_1, \ldots, e_r\}$ are pairs of output and input ports $< a.p^o, a'.p^i >$, where $a, a' \in N$. We can call w a *well-formed workflow* iff

1. All the required input ports p^i ($\neg p^i.opt$) are connected by and edge: $\forall A =< in, out, f, cp > \in N, \forall p^k \in A.in : \neg p^k.opt \Rightarrow \exists e =< p^*, p^k > \in E$

2. All edges involve compatible ports:

$$\forall e =< p^i, p^o >\in E, p^i \rhd p^o.$$

The execution of a well-formed workflow is performed by an *engine* as a sequence of steps. Each step executes all *executable actions* a, which are those whose input ports or configuration parameters cp's have input values available. Input ports are fed by the values prompted on output ports of executed actions, while configuration parameters are provided by users. To this aim, the engine relies on a memory that is a $< key, value >$ map keeping track, for all actions in each instant of the execution, which are the values available for output ports and for configuration parameters. To this aim, the execution of an action a updates the memory with $< key, value >$ entries relative to all output ports in $a.out$.

Definition 5 (Memory) A memory M is defined a set of $< key, value >$ pairs that keeps the status of execution of all workflows w. Keys are of two kinds: action output ports ($w.a.out.p^o$) or action configuration parameters ($w.a.cp.l$). The former are yielded by the execution of actions, the latter are added to M by the researchers.

Definition 6 (Executable Action) Given a workflow $w = (N, E)$, an action $w.a =< in, out, f, cp >$ in w is *executable* according to a given memory M ($w.a \oplus M$) if and only if:

$$\forall l \in w.a.cp : M(w.a.cp.l) \neq \perp)$$

and

$$\forall p^i \in w.a.in. < p^i, p^o >\in w.E : M(w.a.out.p^o) \neq \perp$$

and

$$w.a.executed = false$$

An engine is the agent responsible for the execution of a well-formed workflow w in a memory M.

Definition 7 (Engine) Given a memory M, and a workflow $w = (N, E)$ we define:

– The workflow *starting actions* (nodes without input ports)

$$SA(w) = \{a \in w.N | a.in = \emptyset\}$$

– The workflow *closing actions* (nodes without output ports)

$$CA(w) = \{a \in w.N | a.out = \emptyset\}$$

– The workflow *executable actions* at a given step of execution

$$EA_M(w) = \{a \in w.N | a \oplus M\}$$

– The *execution of an action* $a \in EA_M(w)$ (output values generated by $a.f$)

$$ActionExec_M(a) =$$
$$\{< a.out.p, value > | f \text{ generates } < a.out.p, value >\}$$

and sets $a.executed = true$.

An engine executes a well-formed workflow w in the scope of a given M, inductively, by executing at each step all possible executable actions, updating M accordingly and moving to the next step. Intuitively the execution starts when $EA_M(w)$ equals $SA(w)$ and closes when the engine executes the last set of actions $EA_M(w) = CA(w)$. To this aim an engine defines an execution function $WorkflowExec_M(w)$ that executes an entire workflow w in the context of memory M as follows:

$$WorkflowExec_M(w) = \begin{cases} WorkflowExec_{\bar{M}}(w), & EA_M(w) \neq \emptyset \\ M, & \text{otherwise} \end{cases} \quad (1)$$

where $\bar{M}$ is obtained as

$$\bar{M} = M \cup \bigcup_{a \in EA_M(w)} ActionExec_M(a)$$

5 HyWare platform

5.1 Architecture

HyWare is an hybrid workflow language whose execution environment consists of a platform designed to be integrated as an overlay of any existing research e-infrastructure (RI). The HyWare platform provides i the primitives and recommendations to instruct RI researchers at encoding their "RI actions" in terms of HyWare action classes, and ii the user interfaces to build, test, share HyWare workflows build out of actions. The HyWare platform's architecture consists of three main components (see Figure 3):

User Interface. The User Interface offers to users the functionalities required to i create a workflow in terms of sequences of HyWare actions as made available by the *registry* component, ii discover existing workflows in the registry component, iii execute workflows via the *workflow engine* component.

Registry. The registry is the catalogue of all available HyWare classes, which can be *human* actions or *machine* actions, and also the place where workflows created by users can be deposited for discovery and sharing;

Workflow Engine. The engine offers the functionality to execute a hybrid workflow as a result of the execution of individual machine actions (interaction with third-party services) and human actions it contains.

In the following we shall describe the details of these components and how they are supposed to interact.

5.1.1 User interface

HyWare's user interface supports the user with all functionalities required by a workflow management system. In particular, it enables:

1. Management of "my workflows": creation, update, execution of workflow templates and workflow instances.
2. Discovery of workflow templates and relative instances.
3. Monitoring of workflows: execution history, notifications, etc.
4. Publishing of workflows: DOI minting, linking with publications, making workflow reproducible (if action types make it possible).

Figure 2 shows a mock of the HyWare user interface, with a snapshot taken during the execution of a human action in a workflow. It has divided into two main area. The topmost area depicts the workflow steps by clearly highlighting the already performed actions, the current action and the actions to be performed afterward. The central part contains information about the action currently being executed including a progress bar. The user interfaces require interaction with all platform components, as shown in Fig. 3.

5.1.2 Registry

As mentioned above, the registry is the place where the available human and machine *action classes* are described and the place where workflow templates and workflows are deposited for sharing:

Action classes. An action class is characterized by the following properties: a (unique) *name*, a *description*,the types and location of *input* ports and output ports (as defined in Sect. 4), and a set of *configuration parameters* required to properly execute the action. When the action is machine-actionable, the action class includes the name of the actuator method implementing the business logic of the action and locally deployed within the engine. The user interface allows users to build workflows by searching, selecting and connecting compatible action classes.

Workflow template. A workflow template is a valid combination of action classes, properly pipe-lined in respect of their input and output port types compatibility. Workflow templates were not formally introduced, as they were not necessary to define the semantics of the HyWare engine, but are technically useful to have to support users with an effective workflow management suite. A workflow template specifies how the ports of two subsequent actions a_1 and a_2 are related to each other, identifying for each input port $a_2.p^i$ a relative input–output channel $e =< a_1.p^o, a_2.p^i >$. It specifies the overall logic behind an experiment, in terms of the list of actions required to perform it, still it cannot be executed since it misses a specific configuration of such actions, to be provided by means of the relative configuration parameters.

Workflow. A workflow instance, as defined in Definition 4, is obtained from a workflow template by instantiating all its action classes with the respective configuration parameters, to be provided by the user at workflow configuration time.

Workflow executions. Once completed, the information relative to workflow instance executions are stored in the registry. Information such as initial workflow (actions) parameters, input and output port values for the actions, start time, end time, invoking user, are preserved in the registry. Such *provenance* information is crucial to ensure reproducibility of an experiment and support good Open Science practices.

5.1.3 Workflow engine

The workflow engine implements the logic required to execute a workflow template, hence the relative sequences of actions. As such, its main functions are:

- Executing action instances;
- Enabling parameters passing between different action instances;
- Keeping track of the executions of all workflows.

To this aim, as shown in Figure 3, the engine includes the following components: a *workflow interpreter*, a pool of *actuators* (one for each action class), a *memory*, a *type framework* and a *workflow execution store*.

Action actuators. An actuator is associated with a machine action class. The execution of an action consists in the execution of the actuator which may in turn wrap the invocation of a remote service or simply execute local code. For example, scientists willing to integrate a service so as to make it executable as a machine action in a HyWare workflow, must

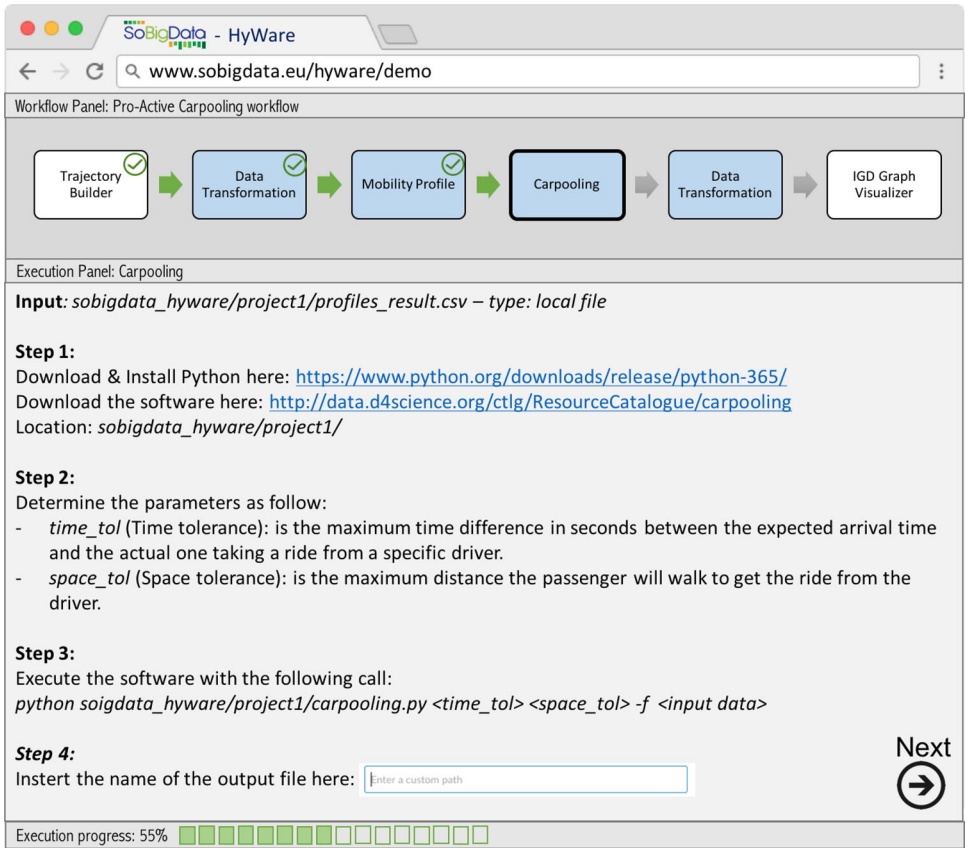

Fig. 2 An example of HyWare web GUI considering the execution of a human action

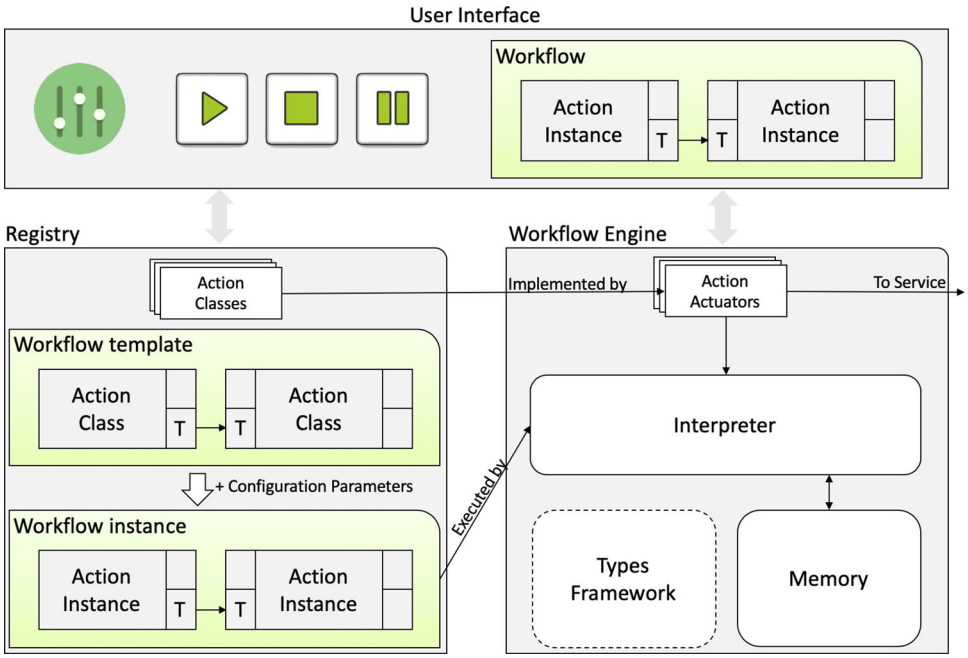

Fig. 3 Registry and workflow engine

therefore define the relative action class (name and properties) and plug-in the actuator required to:

1. Collect all input parameters: parameters are partly from the action instance configuration parameters and partly from the workflow execution environment (see memory below), namely the input ports of the action;
2. Return the results of the invocation: package the results in order for them to be available via the output ports (see memory below) to the subsequent action instance execution.

Memory. The memory component (see Definition 5) is required to ensure correct input–output parameter passing between compatible action ports during the execution of a workflow instance. It acts like a cache of parameter values (or URLs to such values) that the workflow engine relies on to temporarily store the results of an action and feed it to the subsequent one. The value returned on an output port p_o of an action a in a workflow instance w is stored in the memory in the respective entry identified by $(w.a.p_o)_t$, where t is the time-stamp relative to the execution time of the workflow w. The time-stamp is required to distinguish among different executions of the same workflow instances.

The memory supports the interpreter at (*i*) understanding whether or not an action instance can be executed by verifying the availability of all values required by its input ports, and (*ii*) fetching the values to be passed as input to an action and (*iii*) caching the values returned by the output ports of an action after its execution.

Interpreter. Given a workflow instance w in the registry, the user interface lets the user execute either manually or automatically its actions, by interacting with the interpreter component. In order to visualize the current status of an action a, the UI invokes the interpreter that fetches the values relative to (*i*) the configuration parameters available in w for a and to (*ii*) the input parameters for the p_i's of a from the memory component. The user can now execute the action: for human-executable actions the user manually follows the instruction using the values, then provides the values required for the output ports of a and presses the "action complete" button, which stores the output values in memory; for machine-executable actions the user presses the "execute action" button, which causes the interpreter to invoke the actuator relative to the action class of a. The actuator, given the values fetched by the interpreter, executes the action and returns the values to be associated with the output ports in memory.

*Port types framework.*The definition of action classes (Definition 1) in the registry defines the input and output ports for each class as quadruples $< name, type, loc, opt >$. As in typed programming languages, HyWare types suggest to users defining a workflow template (and to validation algorithms checking the correctness of a workflow template) when two actions in a pipeline have input and output ports that are compatible with each other and if a channel can be established between an input and an output port. On the other hand, the rigorousness of types, i.e., the guarantee that an action produces or can interpret values of a given type, is not responsibility of the interpreter, but of the scientists realizing the actions and/or the relative actuators. In other words, the interpreter is not aware and has not control over the effective domain, or exchange format, or internal representation of the values of a type.

In all cases the values returned by an output port are strings to be passed to the next associated input port and be properly interpreted by the relative action actuator.

Users may chose to offer classes that enable management of values of a given types with a custom location. Technically, such classes must implement a set of mandatory and optional methods, standardizing the way values can be accessed, created, updated, searched, preserved for reproducibility and reuse, and visualized in preview in the UI. The resulting type framework enables: (*i*) other users may create actions and relative implementations that operate on such values, and (*ii*) the user interface can activate functionalities to preview input and output values for such types, as well as creating "research objects" to enable the reproducibility of a workflow.

Such high-level types represent a reasonable equilibrium between enabling static control, supporting users with workflow language tools, and at the same time let the user decide to which extent they are willing to integrate their actions into the language.

5.2 Implementing HyWare via KNIME

The HyWare's architecture shown in Figure 3 identifies the key elements for the construction of a platform implementing HyWare's framework and language. In this section we suggest how such elements could be implemented by means of the KNIME platform and relative run-time support. Figure 4 depicts how the reference architecture described in Sec. 5.1 can be implemented by using KNIME [6].

KNIME is an Analytics Platform for data-driven workflows widely used in the community of data analysis. The platform is open source and has a big community of data scientists contributing to the development of different tools.

KNIME workflows are sequences of pipelined KNIME nodes (an example is discussed in Sec. 6). Nodes are the smallest processing unit in KNIME, each node dedicated to perform a specific task (e.g., filtering data rows, training a set, applying a model). Nodes perform a task by executing the relative business logic, given a set of input parameters, which they can collect from their input ports or from a manual configuration. Each node has a state, indicated by the traffic light below the node:

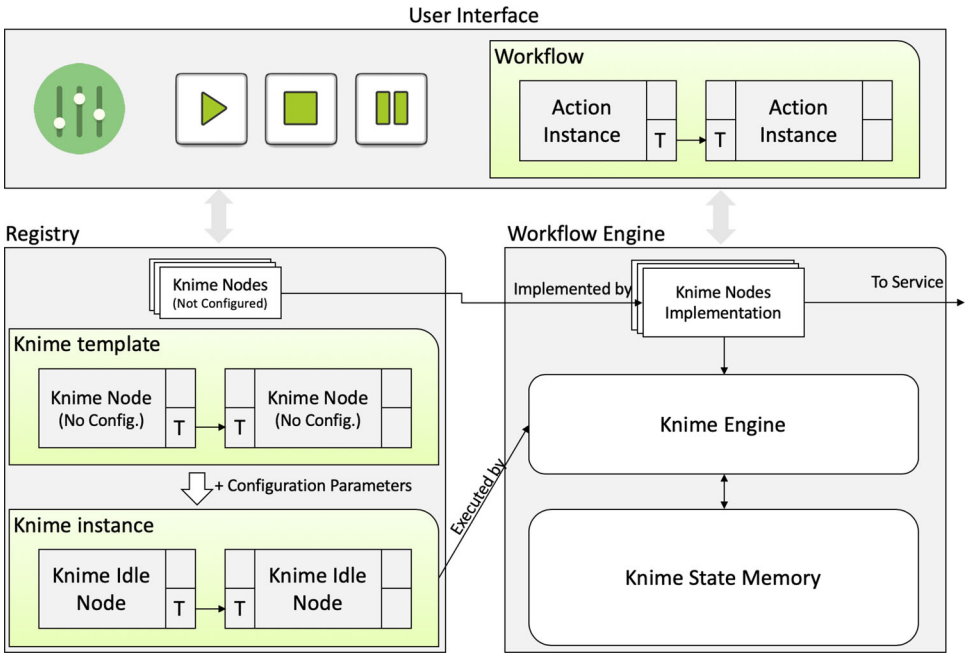

Fig. 4 Registry and workflow engine: KNIME implementation

– Red light ("not configured"): the node has been created, but it is not ready for execution, i.e., missing input parameters;
– Yellow light ("idle"): the node is ready to execute, so configured, but has not yet performed its task;
– Green light ("executed"): the node has executed its task successfully;
– Red light with a cross: ("error"): the node has executed its task with an error.

Similarly to HyWare, KNIME workflows are sequences of nodes, where nodes are connected to each other via their input and output ports. KNIME workflows are graphically specified as shown in Figure 5; workflows and nodes are graphic equivalents of scripts and instructions. The execution of nodes sets the values of the relative output ports, which can be used to feed the input ports of subsequent nodes in the workflow to enable their execution.

New nodes can be introduced in KNIME to perform new tasks by simply implementing a given set of classes and closely following a well-established and clear documentation. Node business logic can be a call to a local procedure (e.g., R, Phyton) or to a remote call to a service embedding any other framework (e.g., SQL databases, MapReduce Hadoop, web services).

In summary, KNIME has addressed several technological challenges mandatory for the implementation of a general-purpose workflow engine, which can be re-used to implement a language of a higher level of abstraction such as HyWare. More specifically:

– HyWare Machine Actions can be implemented as KNIME Nodes with a red light traffic light;
– HyWare Manual Actions can be implemented by introducing a specific family of KNIME nodes, whose execution interacts with the HyWare user interface (notification mode) to enable the visualization of the description of the specific manual action and allow to user to progress with the workflow;
– HyWare Action input/output ports are implemented by structuring the bare textual values in KNIME input/output ports to model ports as quadruples: *name*, *location*, *type*, *value*;
– HyWare Workflow templates are KNIME workflows whose nodes are "not configured";
– HyWare Workflow instances are KNIME workflows whose nodes are "idle."

To complete the implementation specific libraries and structures are required to (*i*) model HyWare ports (quadruples) by means of KNIME ports, (*ii*) model the HyWare memory (i.e., the same action across different workflows should have different ports identifiers) in terms of the KNIME shared memory and (*iii*) to enable the name-based typing of input and output ports supported by HyWare. Similarly, the KNIME classes to be implemented for the realization of a node should make use of HyWare port libraries in order to properly access the actual values needed for their task execution, now part of a quadruple in a HyWare memory.

The HyWare user interface would support users in (*i*) the construction of valid HyWare workflow templates which are in turn stored in the registry as KNIME workflows and (*ii*)

in the execution of such workflows via the KNIME workflow engine. The user interface should fire the execution of a KNIME workflow and be ready to "listen" to any notification that executed workflows might send in correspondence of the execution of a manual action.

6 A real-case scenario for HyWare: the SoBigData e-infrastructure

SoBigData.eu [16] is an European Commission project whose goal is to create the Social Mining & Big Data Ecosystem, i.e., a research infrastructure (RI) providing an integrated ecosystem for ethic-sensitive scientific discoveries and advanced applications of social data mining on the various dimensions of social life, as recorded by "big data." SoBigData.eu is opening up new research avenues in multiple research fields, including mathematics, ICT, and human, social and economic sciences, by enabling easy comparison, reuse and integration of state-of-the-art big social data, methods and services, into new research. Although SoBigData.eu is primarily aimed at serving the needs of researchers, the openly available datasets and open-source methods and services provided by the new research infrastructure will also impact industrial and other stakeholders (e.g., government bodies, non-profit organizations, founders, policy makers).

SoBigData aims at realizing a homogenous e-infrastructure by "gluing together" tools that scientists and practitioners have been realizing in full autonomy and without relying on common interoperability agreements. To this aim, SoBigData e-infrastructure is being realized following an "open ecosystem approach" having the e-infrastructure platform d4science.org as pivot [3] is used. The D4Science platform supports an advanced notion of Virtual Research Environments (VREs), intended as innovative, web-based, community-oriented, comprehensive, flexible and secure working environments conceived to serve the needs of nowaday scientific investigations [2,7]. The implementation and operation of such challenging and evolving working environments largely benefits from and complements the offering of research infrastructures. Via the pivotal infrastructure it is possible to integrate heterogeneous community tools and offering them to scientists in dedicated VREs each tailored to satisfy the needs of a designated community. Clearly this represents a challenge, with the pre-requisites described in previous sections: absence of common standards and protocols, variety in technologies and usage modes, inability or unwillingness to change technology. D4Science acts as the technological bridge via which such tools can be registered, discovered and used by scientists, respecting the will or possibility of the owners of the tools to integrate them fully (e.g., autonomic execution), partly (e.g., web interface integration) or not integrating them in the e-infrastructure (e.g., download and install). In addition, scientists can also benefit from advanced collaborative services: (*i*) a VRE workspace, organized as a shared file system, which allows data to be moved between different tool-based actions, (*ii*) a data analytics platform benefiting from a distributed and multi-tenant computing infrastructure oriented to provide scientists a broad variety of algorithms and methods as well as other kind of web-based resources and (*iii*) collaboration-oriented facilities enabling scientists to publish research results with the possibility to add comments on them in a social-network fashion.

SoBigData scientists can today integrate their tools for VRE-integrated reuse, but cannot represent sequences of actions as a workflow, in order to share it and reproduce it. Equipping SoBigData VREs with HyWare allows scientists to attach to a specific result the entire process used to obtain it. This makes the environment evolve into a living laboratory, which contains not only the methods and the results but also the experience of the researcher in using them in and compose analytic process with it.

In the following sections we shall introduce the classes of actions characterizing SoBigData tools and showcase the usage of HyWare to represent the implementation of the analytic workflow called City of Citizens defined in Example 1.

6.1 SoBigData tools and HyWare actions

In this section we describe the SoBigData.eu infrastructure action classes and instances of such classes, i.e., actions, as described and exemplified in Section 2. Moreover, we shall describe how they are made available in the VREs. As described above the VRE is a working environment tailored to serve the needs of a specific research scenario. The underlying D4Science e-infrastructure platform allows SoBigData scientists to (i) integrate and then register resources, i.e., tools and products (e.g., datasets), to the SoBigData infrastructure and (ii) build VREs as sets of such resources, to support the specific needs of a group of scientists. Specifically, tool resources comprehend and entail the following classes of HyWare actions:

- *Method invocation*: methods (e.g., Java methods, R algorithms, Python methods) to be executed by the D4Science processing engine, integrated based on OGC Web Processing Service (WPS);
- *Software* to be downloaded and executed locally;
- *Web (REST) services* for remote invocation;
- *Web applications*, offering WebUI-accessible functionalities.

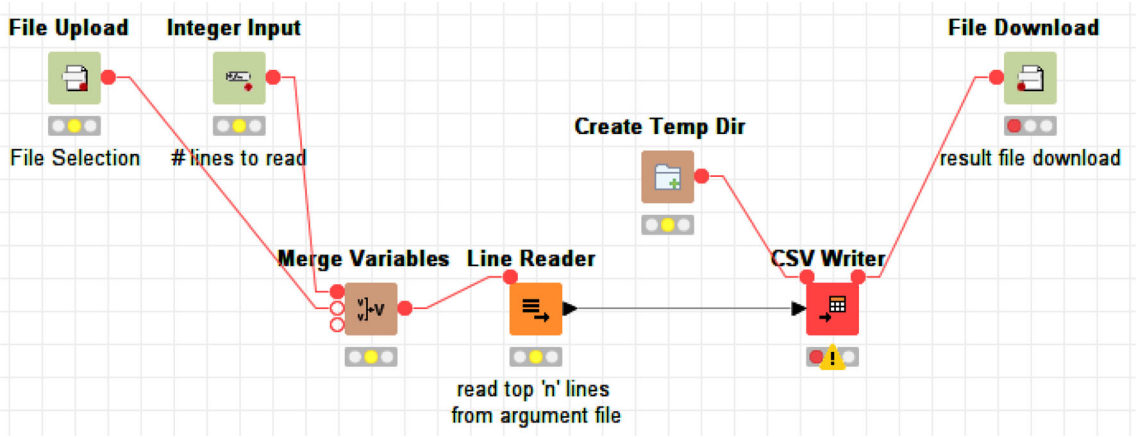

Fig. 5 Example of KNIME workflow

6.2 Implementing the carpooling application

Now that the HyWare is described in details, we recall the example 1 to show how it is implemented in the system. We complete this use case providing the definition of some actions using the HyWare definitions introduced in Sect. 4. The defining is not related to the fact that an action is performed by a human or automatically by the engine. In the next code we codify two representative actions of the workflow in Figure 1.
The first action is the **Trajectory Builder** action that it is automatically executed by system only by setting its parameters.

$$
\begin{aligned}
a_{tb} &= < in = \{Database\_table, \uparrow, true\},\\
out &= \{Database\_table, \uparrow, true\},\\
f_{WS} &= trajectory\_builder(),\\
cp &= \{max\_time\_gap, min\_space\_gap\} >
\end{aligned}
$$

This action requires a database connection to a table as input, executes as web service the function *trajectory_builder()* using the values associate to max_time_gap, min_space_gap as parameters. With this parameter the function separates the original GPS observation sequences into meaningful trajectories cutting where the user remains "static." The result of this computation is stored in the database table (which is the output).

Then an export of the data from the database to a specific data format required by the Mobility Profile is needed, this kind of transformation is not specified in the HyWare core as standard translation, for this reason the researcher created a new Human action:

$$
\begin{aligned}
a_{ex} &= < in = \{Database\_table, \uparrow, true\},\\
out &= \{Local\_file, \downarrow, true\},\\
f_{WS} &= export\_trajectories(),\\
cp &= \{\} >
\end{aligned}
$$

where $export\_trajectories()$ is a function showing the steps needed. The User then is guided through the process:

step 1: Check PgAdmin software in your local computer, if not present download and install it from https://www.pgadmin.org/;
step 2: Connect to the database with the following credentials and locate the table of trajectories: [input];
step 3: Click on the table and select "export table";
step 4: Go to "Dump option1" and select "only data";
step 5: Go to "Dump format" and de-select "use quotation for strings";
step 6: Go to "General" and insert a name for the destination file;
step 7: Insert the name of the exported file here: [output].

Here the [input] is the information received in the input port of the action referring to the database connection and table; moreover, at the end of the process the user must specify the name of the local file exported in the [output] field. The next action is the **Mobility Profile** defined as:

$$
\begin{aligned}
a_{de} &= < in = \{Local\_file, \downarrow, true\},\\
out &= \{Local\_file, \downarrow, true\},\\
f_{WS} &= download\_execute,\\
cp &= \{Execution\_string,\\
&\quad Param\_description, Url\_software\} >
\end{aligned}
$$

where the user is requested to download, install and execute a software on a file. This action is very common, and in

this case, the framework provides a generic action which is suitable for many cases. The general description of the steps is:

step 1: Download software here: [Url_software]
step 2: Determine the parameters as follows: [Param_description]
step 3: Execute the software with the following call: [Execution_string] where the input file is: [input]
step 4: Insert the name of the results file here: [output]

where [Url_software] is where the software can be downloaded, [Param_description] is the description of the parameters for the call of that specific software, i.e., the minimum frequency as the minimum number times a user must follow the same trajectory to be considered systematic trip and the distance tolerance to consider two trajectories equal. Finally the [Execution_string] is the actual string to use to call the software:

$$mobilityprofiler\,[min\_freq]\,[dist\_tol]\; -\; f\,[inputfile]\; -o\,[outputfile]$$

As before the resulting file must be inserted in the output field.

The same action with a different values in the parameters cp is used for the carpooling already shown in fig.2. The resulting systematic movements are matched, and the result is stored in a local file.

The next action is another data transformation from a flat text file to a JSON file. The action is defined as follows:

$$\begin{aligned} a_{de} &= < in = \{Local\_file, \downarrow, true\},\\ out &= \{Local\_file, \downarrow, true\},\\ f_{WS} &= csv\_to\_JSON,\\ cp &= \{\} > \end{aligned}$$

Again the csv_to_JSON function is a human action explaining how to transform the data and require that the result is stored in a file specified as [output].

The last step is an automatic action which takes the local file and upload it in the HyWare workspace to be used in the call of a web page for the visualization of the graph.

$$\begin{aligned} a_{de} &= < in = \{Local\_file, \downarrow, true\},\\ out &= \{\},\\ f_{WS} &= IGD\_Visualizer,\\ cp &= \{\} > \end{aligned}$$

The result of the last action is the visualization of the IGD web application interface with the carpooling matching graph. This example highlights the ability of HyWare to represent processes performed by the analysts, where processes are integrating human and machine-executable actions to flexibly support a degree of reproducibility of the results in highly heterogeneous scenarios.

7 Conclusions

Open Science principles have became relevant in many research sectors, and concepts such as "reproducibility of science" and "transparent assessment" are now central for every research infrastructure. This paper shows that in research infrastructure environments, researchers carry out their scientific processes in terms of sequences of actions called workflows. The generated workflows are themselves products of the researchers.

In this work, we outlined why it is complex managing all the cases inside a research infrastructure since the tools (actions) invocable inside a process are different for nature (e.g., web services or external web applications invoked via API), and they can also include human interaction (e.g., freely download-able software for local execution). For this reason, we proposed the HyWare framework and platform for the representation and reproducibility of hybrid workflows in highly heterogeneous e-infrastructures. The HyWare framework allows scientists to describe the classes of actions they are accustomed to perform, by abstracting over the execution of web services and over so-called human actions. The HyWare platform offers user interfaces to support the scientists at constructing a workflow out of the available action classes and make it available for others to discovery.

Despite the degree of heterogeneity of "actions," e-infrastructures equipped with HyWare embrace Open Science principles. They allow scientists to share (publish) workflows, repeat and validate science. We exemplified the usage of HyWare in the real context of the SoBigData e-infrastructure and illustrated a possible implementation of the platform via the KNIME framework and engine.

The workflow engine is currently being implemented in the context of D4Science so as to make it available to all e-infrastructures it supports (including SoBigData). This will enable us to have experience for evaluating the engine (and the language) on a large set of real examples created by the research infrastructure users. The implementation will demonstrate definitively how useful your approach is, and what are the advantages when it is used in practice.

Acknowledgements This work is supported by the European Community's H2020 Program under the scheme "INFRAIA-1-2014-2015: Research Infrastructures," Grant agreement n. 654024 "SoBigData: Social Mining & Big Data Ecosystem" (http://www.sobigdata.eu).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Amstutz, P., Crusoe, M.R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., Scales, M., Soiland-Reyes, S., Stojanovic, L.: Common workflow language, v1.0. Specification (2016). https://doi.org/10.6084/m9.figshare.3115156.v2
2. Assante, M., Candela, L., Castelli, D., Cirillo, R., Coro, G., Frosini, L., Lelii, L., Mangiacrapa, F., Marioli, V., Pagano, P., Panichi, G., Perciante, C., Sinibaldi, F.: The gCube system: delivering virtual research environments as-a-service. Future Gener. Comput. Syst. **95**, 445–453 (2019). https://doi.org/10.1016/j.future.2018.10.035
3. Assante, M., Candela, L., Castelli, D., Cirillo, R., Coro, G., Frosini, L., Lelii, L., Mangiacrapa, F., Pagano, P., Panichi, G., Sinibaldi, F.: Enacting open science by D4Science. Future Gener. Comput. Syst. **101**, 555–563 (2019). https://doi.org/10.1016/j.future.2019.05.063
4. Bartling, S., Friesike, S.: Towards another scientific revolution. In: Opening Science, pp. 3–15. Springer, Berlin (2014)
5. Becker, J., Rosemann, M., von Uthmann, C.: Guidelines of business process modeling. In: Business Process Management, pp. 30–49. Springer, Berlin (2000). https://doi.org/10.1007/3-540-45594-9_3
6. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., Wiswedel, B.: Knime-the Konstanz information miner: version 2.0 and beyond. SIGKDD Explor. Newsl. **11**(1), 26–31 (2009). https://doi.org/10.1145/1656274.1656280
7. Candela, L., Castelli, D., Pagano, P.: Virtual research environments: an overview and a research agenda. Data Sci. J. **12**, GRDI7GRDI75–GRDI81 (2013). https://doi.org/10.2481/dsj.GRDI-013
8. Candela, L., Giannotti, F., Grossi, V., Manghi, P., Trasarti, R.: Hyware: a hybrid workflow language for research e-infrastructures. D-Lib Magazine (2017). https://doi.org/10.1045/january2017-candela
9. Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., Hinsen, K., Larmande, P., Bras, Y.L., Lemoine, F., Mareuil, F., Ménager, H., Pradal, C., Blanchet, C.: Scientific workflows for computational reproducibility in the life sciences: status, challenges and opportunities. Future Gener. Comput. Syst. **75**, 284–298 (2017). https://doi.org/10.1016/j.future.2017.01.012
10. Coro, G., Panichi, G., Scarponi, P., Pagano, P.: Cloud computing in a distributed e-infrastructure using the web processing service standard. Concurrency and Computation: Practice and Experience **29**(18), e4219. https://doi.org/10.1002/cpe.4219. E4219 cpe.4219
11. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., da Silva, R.F., Livny, M., Wenger, K.: Pegasus, a workflow management system for science automation. Future Gener. Comput. Syst. **46**, 17–35 (2015). https://doi.org/10.1016/j.future.2014.10.008
12. Filgueira, R., Atkinson, M., Bell, A., Main, I., Boon, S., Kilburn, C., Meredith, P.: Escience gateway stimulating collaboration in rock physics and volcanology. pp. 187–195 (2014). https://doi.org/10.1109/eScience.2014.22
13. Garijo, D., Alper, P., Belhajjame, K., Corcho, O., Gil, Y., Goble, C.: Common motifs in scientific workflows: An empirical analysis. Future Generation Computer Systems **36**, 338 – 351 (2014). https://doi.org/10.1016/j.future.2013.09.018. Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications
14. Garijo, D., Gil, Y., Corcho, O.: Abstract, link, publish, exploit: An end to end framework for workflow sharing. Future Generation Computer Systems **75**, 271–283 (2017). https://doi.org/10.1016/j.future.2017.01.008
15. Giannotti, F., Trasarti, R., Bontcheva, K., Grossi, V.: Sobigdata: Social mining & big data ecosystem. In: Companion Proceedings of The Web Conference 2018, WWW '18, pp. 437–438. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE (2018). https://doi.org/10.1145/3184558.3186205. https://doi.org/10.1145/3184558.3186205
16. Giannotti, F., Trasarti, R., Bontcheva, K., Grossi, V.: Sobigdata: Social mining & big data ecosystem. In: Companion of The Web Conference 2018 on The Web Conference 2018, pp. 437–438. International World Wide Web Conferences Steering Committee (2018)
17. Goble, C., Cohen-Boulakia, S., Soiland-Reyes, S., Garijo, D., Gil, Y., Crusoe, M.R., Peters, K., Schober, D.: Fair computational workflows. Data Intell. **2**(1–2), 108–121 (2020). https://doi.org/10.1162/dint_a_00033
18. Goecks, J., Nekrutenko, A., Taylor, J.: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. **11**(8), R86 (2010). https://doi.org/10.1186/gb-2010-11-8-r86
19. Kougka, G., Gounaris, A., Simitsis, A.: The many faces of data-centric workflow optimization: a survey. Int. J. Data Sci. Anal. **6**(2), 81–107 (2018). https://doi.org/10.1007/s41060-018-0107-0
20. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurr. Comput. Pract. Exp. **18**(10), 1039–1065. https://doi.org/10.1002/cpe.994
21. LeVeque, R.J., Mitchell, I.M., Stodden, V.: Reproducible research for scientific computing: tools and strategies for changing the culture. Comput. Sci. Eng. **14**(4), 13–17 (2012). https://doi.org/10.1109/MCSE.2012.38
22. Liew, C.S., Atkinson, M.P., Galea, M., Ang, T.F., Martin, P., Hemert, J.I.V.: Scientific workflows: moving across paradigms. ACM Comput. Surv. (2016). https://doi.org/10.1145/3012429
23. Llorà, X., Ács, B., Auvil, L.S., Capitanu, B., Welge, M.E., Goldberg, D.E.: Meandre: Semantic-driven data-intensive flows in the clouds. In: 2008 IEEE Fourth International Conference on eScience, pp. 238–245 (2008). https://doi.org/10.1109/eScience.2008.172

24. Marru, S., Gunathilake, L., Herath, C., Tangchaisin, P., Pierce, M., Mattmann, C., Singh, R., Gunarathne, T., Chinthaka, E., Gardler, R., Slominski, A., Douma, A., Perera, S., Weerawarana, S.: Apache airavata: A framework for distributed applications and computational workflows. In: Proceedings of the 2011 ACM Workshop on Gateway Computing Environments, GCE '11, pp. 21–28. ACM, New York, NY, USA (2011). https://doi.org/10.1145/2110486.2110490
25. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17), 3045 (2004). https://doi.org/10.1093/bioinformatics/bth361
26. Plankensteiner, K., Prodan, R., Janetschek, M., Fahringer, T., Montagnat, J., Rogers, D., Harvey, I., Taylor, I., Balaskó, Á., Kacsuk, P.: Fine-grain interoperability of scientific workflows in distributed computing infrastructures. J. Grid Comput. **11**(3), 429–455 (2013). https://doi.org/10.1007/s10723-013-9261-8
27. Roure, D.D., Goble, C., Stevens, R.: The design and realisation of the experiment my virtual research environment for social sharing of workflows. Future Gener. Comput. Syst. **25**(5), 561–567 (2009). https://doi.org/10.1016/j.future.2008.06.010
28. Schaduangrat, N., Lampa, S., Simeon, S., Gleeson, M.P., Spjuth, O., Nantasenamat, C.: Towards reproducible computational drug discovery. J. Cheminformatics **12**(1), 9 (2020). https://doi.org/10.1186/s13321-020-0408-x
29. Schiermeier, Q.: Europe is a top destination for many researchers. Nature **569**(7757), 589–591 (2019). https://doi.org/10.1038/d41586-019-01570-3
30. Shaon, A., Callaghan, S., Lawrence, B., Matthews, B., Woolf, A., Osborn, T., Harpham, C.: A linked data approach to publishing complex scientific workflows. In: 2011 IEEE Seventh International Conference on eScience, pp. 303–310 (2011). https://doi.org/10.1109/eScience.2011.49
31. Stodden, V., Guo, P., Ma, Z.: Toward reproducible computational research: an empirical analysis of data and code policy adoption by journals. PLoS ONE **8**(6), 1–8 (2013). https://doi.org/10.1371/journal.pone.0067111
32. Teytelman, L., Stoliartchouk, A., Kindler, L., Hurwitz, B.L.: Protocols io virtual communities for protocol development and discussion. PLOS Biol. **14**(8), 1–6 (2016). https://doi.org/10.1371/journal.pbio.1002538
33. Weske, M.: Business process management architectures. In: Business Process Management pp. 333–371. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28616-2_7
34. Wilsdon, J., de Rijcke, S.: Europe the rule-maker. Nature **569**(7757), 479–481 (2019). https://doi.org/10.1038/d41586-019-01568-x
35. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., Bhagat, J., Belhajjame, K., Bacall, F., Hardisty, A., Nieva de la Hidalga, A., Balcazar Vargas, M.P., Sufi, S., Goble, C.: The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Research **41**(W1), W557–W561 (2013). https://doi.org/10.1093/nar/gkt328. http://dx.doi.org/10.1093/nar/gkt328
36. Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., Wilde, M.: Swift: Fast, reliable, loosely coupled parallel computation. In: 2007 IEEE Congress on Services (Services 2007), pp. 199–206 (2007). https://doi.org/10.1109/SERVICES.2007.63