Linking Data, Services and Human Know-How

Paolo Pareti[1(✉)], Ewan Klein[1], and Adam Barker[2]

[1] University of Edinburgh, Edinburgh, UK
p.pareti@sms.ed.ac.uk, ewan@inf.ed.ac.uk
[2] University of St Andrews, St Andrews, UK
adam.barker@st-andrews.ac.uk

Abstract. An increasing number of everyday tasks involve a mixture of human actions and machine computation. This paper presents the first framework that allows non-programmer users to create and execute workflows where each task can be completed by a human or a machine. In this framework, humans and machines interact through a shared knowledge base which is both human and machine understandable. This knowledge base is based on the PROHOW Linked Data vocabulary that can represent human instructions and link them to machine functionalities. Our hypothesis is that non-programmer users can describe how to achieve certain tasks at a level of abstraction which is both human and machine understandable. This paper presents the PROHOW vocabulary and describes its usage within the proposed framework. We substantiate our claim with a concrete implementation of our framework and by experimental evidence.

1 Introduction

This paper addresses the largely unexplored problem of enabling non programmers to specify the type of behaviour they require from machines, as opposed to being constrained by their pre-defined functionalities. Applications such as IFTTT[1] (If-This-Than-That) have demonstrated that, given the right tools, users are capable of composing different components, such as triggers and actions, to program certain desired behaviours on a system. IFTTT for example, allows users to define conditions, such as "if there is a chance of rain tomorrow", in order to automatically trigger actions, such as "notify me".

At the opposite end of the spectrum from computer programs, human instructions can be seen as ways to "program" human behaviours. Humans are capable of specifying complex workflows based on human actions. Online instructions, such as those available on the wikiHow[2] website, often include different methods, steps, loops, and conditions. The main disadvantage of human instructions is that they are not machine understandable. Consequently, any useful functionality that a human could benefit from while following a set of instructions has to be manually accessed and triggered by the user.

[1] http://ifttt.com/.
[2] http://www.wikihow.com/.

H. Sack et al. (Eds.): ESWC 2016, LNCS 9678, pp. 505–520, 2016.
DOI: 10.1007/978-3-319-34129-3_31

The main contributions of this paper are two. The first is a framework that allows non-programmers to define and execute human-machine workflows. Our approach overcomes the limitations of traditional human instructions by allowing machines to automatically detect when certain actions are needed, and consequently to actively execute them. Section 4 presents the components of our framework and describes how they achieve its objective. An implementation of this framework is described in Sect. 6 and its usability by non-programmer users is evaluated in Sect. 7.

The second contribution is the PROHOW Linked Data vocabulary that we use to represent this knowledge base. This vocabulary not only describes human-machine workflows but can also trigger and describe their execution. Thus Linked Data, in our framework, acts both as a data representation and as a programming language. The details of this vocabulary, along with its intended logical interpretation, are given in Sect. 5.

2 Motivating Example

As a running example, we consider a scenario where a non-programmer human is writing instructions that include some automatable steps. In this scenario, Jane is an employee of a company who is in charge of curating content on the company website. Today Jane has gained access to a text file containing a long list of cities that the company has worked with. Each line of the file contains the name and Wikipedia page of the city in the following format:

```
<London: https://en.wikipedia.org/wiki/London>
```

Jane's job is to display the name and official website of each city as a list on the company's website. To achieve this, she wants to transform every entry in the dataset into an HTML list element in this format:

```
<li>London: https://london.gov.uk/</li>
```

Jane decides to delegate this task (that we will denote t) to John, one of her collaborators, and she gives him these instructions:

> Step 1: Remove < and > from the string of text.
> Step 2: Substitute the wikipedia URL with the official website of the city.
> Step 3: Enclose the string of text with <li> .

Jane notices that some steps in her instructions could be easily automated by a machine. However, Jane's problem is that nobody in her company is a programmer. Consequently, although some of the required functionalities are available in their computers, the whole task might have to be completed manually.

Our proposed solution for Jane's problem is to let her use a system that automatically translates her natural language instructions into machine understandable data and that semi-automatically links them to machine functionalities. This system might analyse the natural language description of steps 1 and 3 and detect that they fall within its capabilities. Given this configuration,

when task t is started with a new string of text as input, the system could immediately execute the first step of the instructions. Then, as soon as the second step is complete, the third step will also be automatically executed.

The system will store all the information about Jane's task as Linked Data. This will allow the system to publish this information to other systems, and maybe discover that there is an external service that can also automate the second step of Jane's instructions. For example, an external service might be able to automate Jane's second step by querying the DBpedia[3] dataset, which contains information about the official websites of a large number of cities.

An important observation that can be derived from this example is that it is inefficient to create ad hoc systems to achieve these types of tasks, especially when they are small in scale, and when they can occur frequently but with variations. For example, other problems might require the same functionalities required by t, but combined together into a different workflow. Instead of creating rigid ad hoc solutions for human-machine collaboration, our approach to automation makes use of simple and generic machine capabilities that can be integrated with several different human-made instructions.

Many such capabilities can be imagined. For example, a trigger capability could instruct a machine to start the task "organise lunch in the park" if the weather is sunny and if no other commitment is scheduled for lunchtime in the user's calendar. If the user decides to do this activity, the calendar might also be updated automatically. Alternatively, a machine could assist the organiser of an event who decides to "send a message" to a large number of invitees. A machine might know two methods to automatically "send a message" to a person: by email and by mobile text. The machine could automatically send the message to all invitees whose email or mobile number is known. The remaining invitees could then be manually contacted by the event organiser using other channels. This scenario highlights the flexibility of human-machine collaboration, since a purely manual approach would be inefficient, and a purely automatic approach would be infeasible. More examples of machine capabilities that are being used by non-programmer users can be found on the IFTTT website.

3 Problem Description

The concept of *tasks* refers to things that can be accomplished. As such, they can be used to define goals, namely things that a person wants to accomplish. The main use of the concept of tasks is to provide a layer of abstraction over the actual *actions* that are performed to accomplish them. The types of tasks that humans can describe is very broad, and when interacting with machines, the level of abstraction at which they are described plays an important role. At the opposite ends of the abstraction spectrum we can find very abstract tasks, such as "Behave well", and very specific tasks, such as "Increment variable X by 1 unit". Typically, machines struggle to understand abstract tasks, while they can often accomplish specific tasks more efficiently than humans. Humans, on the other hand, can easily

[3] http://wiki.dbpedia.org/.

describe tasks in abstract terms, but struggle to define them in a very specific and rigorous way. Tasks that are too abstract for machines to automate, or too specific for humans to describe, are outside the focus of this paper. Our hypothesis is that there is a non-empty intersection T between the tasks that can be easily defined by (non-programmer) humans H, and those that are understandable by machines M. Lacking a previous baseline to compare to, we define a task to be *easily* definable if the majority of (non-programmer) humans can define it at a sufficient level of abstraction for it be machine understandable.

Our objective is to allow (non-programmer) humans to make better use of machine functionalities on the Web by allowing machines to understand which of their services is needed and when. This type of human-machine collaboration can be achieved by having humans and machines interact through a shared knowledge base. To this end, the problem that we address is how to allow a (non-programmer) web user to share (1) human-made instructions and (2) information on the progress made at completing the tasks described in those instructions, in a format that is both human and machine understandable.

4 Methodology

At the core of our approach for allowing human-machine collaboration is a knowledge base that is shared between humans and machines. Humans and machines collaborate with each other by interacting with this knowledge base, as depicted in Fig. 1. The contents of the knowledge base rely on the PROHOW vocabulary. This is a Linked Data vocabulary that we developed for key concepts that occur in human-made instructions (such as steps and requirements) or that describe aspects of their execution (such as information as to which steps have been completed).

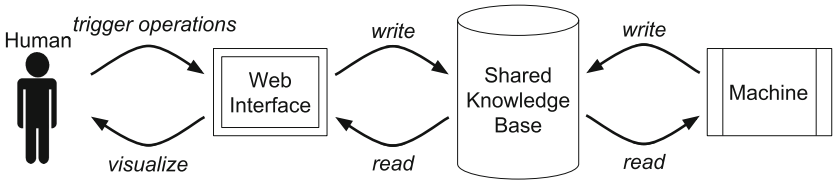

Fig. 1. Schema of the main components of our framework. Humans and machines interact with each other through a shared knowledge base.

In our approach, communication is performed indirectly, by modifying the shared knowledge base. This can be seen as a type of *stigmergy* [6], namely indirect communication through modification of the environment. For example, if a machine wants to communicate to humans (or to other machines) that a particular step has been automated, this will be done by adding this information to the knowledge base. This type of indirect communication avoids the problem of

how to implement direct communication between human and machines. Instead, it casts human-machine collaboration as a knowledge sharing problem, and as such it is amenable to being handled by Semantic Web technologies.

Human and machines interact with the knowledge base in two different ways. Machines can directly access it to query or modify its contents, since the knowledge base is represented as an RDF[4] graph. Moreover, machines can understand this knowledge base by following its logical interpretation, as defined by the PROHOW vocabulary. This logical interpretation allows machines to make inferences over this knowledge base. For example, a machine could infer that a certain task has been implicitly accomplished because all of its steps have been completed, or it could infer that it is not yet time to complete a certain task because some of its requirements are still incomplete. The logical interpretation of the PROHOW vocabulary will follow in Sect. 5.

Humans, on the other hand, interact with this knowledge base through an intuitive Web interface. This interface allows humans to "read" the knowledge base by providing it in a human-readable format. For example, while a machine could query the knowledge base to retrieve all the steps of a given task, a human could visualize the list of steps in an HTML page. This interface also allows humans to "write" to the knowledge base. This can be done, for example, by parsing a user's natural language instructions into RDF, or by interpreting a user action of ticking off a certain step as an indication that the step has been completed. An implementation of this interface will be presented in Sect. 6.

Once humans and machines interact through a shared knowledge base, they can collaborate on the execution of tasks. This collaboration can be divided into three phases: (1) know-how acquisition, (2) know-how linking and (3) execution. We will now describe the typical workflow of our framework across these phases.

4.1 Know-How Acquisition

In the first phase of our framework, know-how is converted into Linked Data. Our hypothesis is that humans can write instructions in semi-structured format. Websites like wikiHow, for example, require users to explicitly divide their instructions into steps, methods and requirements. Further support for our hypothesis is provided by the existence of large instructional websites that contain instructions with this level of structure. Evidence of this has been provided by a large scale conversion of over 200,000 instructions from the wikiHow and Snapguide[5] websites into an RDF format using the PROHOW vocabulary [7].

This existing structure can be extracted and represented in RDF. For example, consider the following natural language instructions for the string transformation task `:t` described in Sect. 2:

```
Step 1: Remove < and > from the string of text.
Step 2: Substitute the wikipedia page of the city in the
    string of text with official homepage of the city.
Step 3: Enclose the string of text with <li> </li>.
```

[4] http://www.w3.org/TR/rdf11-concepts/.
[5] https://snapguide.com/.

Table 1. The RDF namespaces used in this document.

Prefix	Namespace
`prohow:`	http://w3id.org/prohow
`rdfs:`	http://www.w3.org/2000/01/rdf-schema
:	http://example.org/

By exploiting the implicit structure of this text, it is possible to automatically generate the following RDF graph. RDF graphs listed in this paper are serialised in Turtle[6] format, and use the namespaces defined in Table 1.

```
:t prohow:has_step :1, :2, :3 .
:1 rdfs:label "Remove < and > from the string of text.".
:2 rdfs:label "Substitute the wikipedia page of the city in the
    string of text with official homepage of the city.".
:3 rdfs:label "Enclose the string of text with <li> </li>.".
:2 prohow:requires :1 .
:3 prohow:requires :2 .
```

This RDF graph explicitly represents the subdivision of task `:t` into three steps (`:1`, `:2` and `:3`) using the `prohow:has_step` relation. The correct ordering of the steps is specified by the `prohow:requires` relations.

4.2 Know-How Linking

In the second phase of our methodology, an existing set of instructions is linked with automatable functions. For example, we can imagine a machine `:x` capable of removing specific characters from strings of text. This machine can describe its capability in terms of the task it can accomplish. For example, the following RDF graph describes the task `:t1` of "Remove a character from a string of text". This task specifies the requirements `:r1` "The string of text to modify" and `:r2` "The character to remove" which should be known before the task can be automated.

```
:t1 rdfs:label "Remove a character from a string of text".
:r1 rdfs:label "The string of text to modify" .
:r2 rdfs:label "The character to remove" .
:t1 prohow:requires :r1, :r2 .
```

PROHOW allows functionalities to be defined at the input/output level. A functionality `:f` with a set of inputs I and a set of outputs O is described with a set of `prohow:requires` and `prohow:has_method` relations. A `prohow:requires` link from `:f` to one of its inputs `:i` represents the dependency between `:f` and `:i`, thus making sure that the functionality will not be executed before its input is available. A `prohow:has_method` link from an output `:o` to `:f` represents the fact that one way to obtain `:o` is to perform `:f`.

[6] http://www.w3.org/TR/turtle/.

It can be observed that the `prohow:requires` relation can be used to represent both the dependency between (1) a task and an input and (2) a task and a step that needs to be done beforehand. This is a result of the fact that in the domain of human know-how the distinction between actions and objects can be blurred. A cooking recipe, for example, could mention the ingredient "eggs" as an input, or it could mention "get eggs" as one of its steps. In such a situation, the choice of whether to represent something as an object or as an action is arbitrary, and it should not lead to semantically different formalisations. Therefore, the PROHOW vocabulary diverges from typical process formalisations in that it does not enforce a distinction between actions and objects.

Going back to our example, we can imagine a step `:t2` of a procedure that requires the character "<" to be removed from string `:s`. This step can be linked to the functionality offered by machine `:x` with the following triples:

```
:t2 prohow:has_method :t1 .
:t2 prohow:has_constant :c1 .
:c1 rdfs:label "<" .
:r1 prohow:binds_to :s .
:r2 prohow:binds_to :c1 .
```

Once this link has been created, machine `:x` will detect that its capability of accomplishing task `:t1` can also be used to accomplish task `:t2`. When task `:t2` needs to be completed, machine `:x` will try to accomplish it by executing `:t1`. Bindings between tasks can be used to specify which particular parametrisation of a task can be used to accomplish another task. In this scenario, for example, the character to remove `:r2` is bound to the constant "<". These types of links can connect a single machine functionality, or one of its parametrisations, to any number of more abstract tasks that this functionality can accomplish. For example, task `:t1` could be linked to any string modification task that specifies a string and a character to remove from that string.

The discovery of these kind of links can be seen as a form of subsumption matching, since the set of possible ways of accomplishing the more abstract task subsume the set of possible ways of accomplishing the more specific one. This discovery process can be performed in different ways. In a previous experiment, we automated the creation of links between different sets of instructions from wikiHow (in the PROHOW format) using Natural Language Processing and Machine Learning [7]. This showed that creating links between sets of PROHOW instructions can be achieved with high accuracy. Indeed, the number and precision of the discovered links was shown to be superior to the equivalent human-generated HTML links already present in wikiHow. In this paper we consider instead a semi-automated approach. Whenever a set of instructions is created, an artificial system can select from a large number of available resources the ones that seem to be most related to each part of the instructions. Humans will then be asked to verify whether a link should be created or not.

4.3 Execution

When a human or a machine intends to execute a task, an RDF graph is created that declares a new execution of that task. For example, the following triple is sufficient to declare a new attempt `:en` to accomplish task `:t`.

```
:en prohow:has_goal :t .
```

Declaring this intention could be as simple as pressing a "Do it!" button available on the same web page that describes task `:t`.

After the creation of this triple, it is possible to retrieve information about `:en` in order to view (and if necessary modify) the current state of the execution. For a human user, a visualization of `:en` could display, for example, which steps of the procedure have already been completed, and which still need to be completed. We can represent the fact that the execution `:ex1` of the first step `:1` of the instructions `:t` has been completed with the following graph:

```
:ex1 prohow:has_task :1 .
:ex1 prohow:has_result prohow:complete .
:ex1 prohow:has_environment :en .
```

After this information is stored in the shared knowledge base, all the humans and machines collaborating on this task will be able to discover that the first step `:t1` has been completed. They would then infer that it no longer needs to be carried out and could decide to execute the following step instead.

5 Logical Interpretation of the PROHOW Vocabulary

This section describes the logical interpretation of the PROHOW vocabulary that enables machines to understand the shared knowledge base of our framework. The terms of this vocabulary are listed in Table 2, and follow the namespaces described in Table 1.

The most important concept in the PROHOW vocabulary is the concept of *task* (`prohow:task`). The same task can be accomplished multiple times. For example, in the example scenario introduced in Sect. 2, the string transformation task would need to be completed once for each string of text that needs to be transformed. The concept of *environment* (`prohow:environment`) is used to group together all the information about a specific intention to achieve a task. We use the logical expression has_goal(e, y) to indicate that the main task (or goal) of environment e is task y.

With the term *execution* (`prohow:execution`) we refer to a particular attempt to complete a task. A task y is complete (`prohow:complete`) in an environment e if there is an execution i in that environment that has succeeded (Formula 1); it is said to be failed (`prohow:failed`) if that execution has failed instead (Formula 2). Intuitively, the completion of a task means that a satisfactory result has been reached, and there is no need for completing the same task

in the same environment again. If an execution has failed instead, it is possible to attempt another execution of the same task in the same environment.

$$\text{complete}(y, e) \Longleftarrow \exists i.\text{has\_env}(i, e) \wedge \text{has\_task}(i, y) \wedge \text{success}(i) \quad (1)$$

$$\text{failed}(y, e) \Longleftarrow \exists i.\text{has\_env}(i, e) \wedge \text{has\_task}(i, y) \wedge \text{failure}(i) \quad (2)$$

An environment is said to be finished when its goal is complete:

$$\text{finished}(e) \Longleftarrow \exists g.\text{has\_goal}(e, g) \wedge \text{complete}(g, e) \quad (3)$$

Tasks are not only used to denote actions, such as "Remove all the < and > characters from a string of text" but they are also used to refer to objects and data, such as "the string of text" that needs to be modified. In this last case, completing a task means obtaining the object or discovering its value as a variable. The particular object/value associated with a completed execution might be specified using the has_value relation. With the following formula we state that an object-task y has a value of z in environment e if there is a complete execution in that environment that refers to task y and has value z:

$$\begin{aligned} &\text{value}(z, y, e) \Longleftarrow \exists i.\text{has\_env}(i, e) \\ &\wedge \text{has\_task}(i, y) \wedge \text{success}(i) \wedge \text{has\_value}(i, z) \end{aligned} \quad (4)$$

A common property of tasks is the set of requirements that need to be completed before the execution of the task is ready to start. A task y is ready in an environment e if all of its requirements (if any) are complete in the same environment (Formula 5). If a task is not ready, than no attempt at performing it, or any of its sub-tasks, should occur. Intuitively, this means that a task can be started only when all of its requirements are complete. This relation can also be used to order (totally or partially) the steps of a task.

$$\text{ready}(y, e) \Longleftarrow \forall x.\text{requires}(y, x) \rightarrow \text{complete}(x, e) \quad (5)$$

If a task y has at least one step, and all of its steps are complete in an environment e, then task y is also complete in that environment (Formula 6). Intuitively, this means that a task can be accomplished by accomplishing all of its steps.

$$\text{complete}(y, e) \Longleftarrow \exists x.\text{has\_step}(y, x) \bigwedge \forall x.\text{has\_step}(y, x) \rightarrow \text{complete}(x, e) \quad (6)$$

If a method x of a task y is complete in a sub-environment of e, then task y is complete in environment e (Formula 7). Intuitively, this means that a task can be completed by completing any of its methods.

$$\text{complete}(y, e) \Longleftarrow \exists x, a.\text{has\_method}(y, x) \wedge \text{complete}(x, a) \wedge \text{sub\_env}(a, e) \quad (7)$$

Environments that are connected with the sub-environment relation are said to be related environments:

$$\begin{aligned} &\text{related}(a, e) \Longleftarrow \text{sub\_env}(a, e) \vee \text{sub\_env}(e, a) \\ &\text{related}(a, e) \Longleftarrow \exists x.\text{related}(a, x) \wedge \text{related}(x, e) \end{aligned} \quad (8)$$

Unless bindings have been specified, a task that is complete in one environment is not necessarily complete in its related environments. A task x is complete in an environment e if it has a binding with another task y which is complete is an environment a related to e (Formula 9). Values can be shared in a similar way, as defined in Formula 10.

$$\text{complete}(x, e) \Longleftarrow \exists y, a.\text{binds}(x, y) \wedge \text{complete}(y, a) \wedge \text{related}(a, e) \quad (9)$$

$$\text{value}(z, x, e) \Longleftarrow \exists y, a.\text{binds}(x, y) \wedge \text{value}(z, y, a) \wedge \text{related}(a, e) \quad (10)$$

Bindings can be used to choose which tasks and values can be shared between environments. For example, the task "Remove a character from a string of text" could be completed in two different (but related) environments to remove two different characters (e.g. < and >) from the same string of text. In this scenario, the string of text will be shared between the two environments, while the characters to remove will not.

Values that can be shared between unrelated environments are called constants. If a task y has a constant x, then x will be automatically considered accomplished in any environment where y is being executed:

$$\begin{aligned} &\text{complete}(x, e) \wedge \text{value}(x, x, e) \Longleftarrow \exists i.\text{has\_env}(i, e) \\ &\wedge \text{has\_task}(i, y) \wedge \text{has\_constant}(y, x) \end{aligned} \quad (11)$$

It should be noted that the expressiveness of the PROHOW vocabulary goes beyond simple step sequences. In fact, this vocabulary can express all the basic control flow patterns defined by Van der Aalst et al. [11]. For example, tasks are by default non-ordered and any partial ordering of the tasks can be expressed using `prohow:requires` relations. This implements the *sequence*, *parallel split* and *synchronisation* patterns. The `prohow:has_method` relation can describe a choice point where only one out of multiple paths needs to be followed. Other tasks can be made to wait until one such path has been completed. This implements the *exclusive choice* and *simple merge* patterns.

6 Human Interfaces to the PROHOW Vocabulary

Unlike machines, humans cannot directly interact with PROHOW data. Their interactions need to be mediated through human-understandable interfaces that allow users to both visualise and modify this data. For example, such an interface might present entities organised into familiar structures, such as an ordered list of steps or to-do checklists. In order to allow users to modify the data, several functionalities need to be implemented. In particular, users should be able to create new sets of instructions, create links between them, start new executions of a task and update their progress.

As a proof of concept, we have implemented one such interface as an online service[7] which allows users to follow the three main phases defined in Sect. 4.

[7] http://w3id.org/prohow/editor.

Table 2. The concepts and relations of the PROHOW vocabulary.

Concept	Description of the concept
`prohow : task`	a task that can be accomplished
`prohow : execution`	an attempt to perform a task
`prohow : environment`	a collection of executions to achieve a goal
`prohow : complete`	the positive result of accomplishing a task
`prohow : failed`	the negative result of accomplishing a task
Relation	Logical definition (with subject y and object x)
`prohow : requires`	requires(y, x)
`prohow : has_step`	has_step(y, x)
`prohow : has_method`	has_method(y, x)
`prohow : has_task`	has_task(y, x)
`prohow : has_goal`	has_goal(y, x)
`prohow : binds_to`	binds$(y, x) \wedge$ binds(x, y)
`prohow : has_constant`	has_constant(y, x)
`prohow : has_value`	has_value(y, x)
`prohow : has_result`	success(y) if x is `prohow : complete` failure(y) if x is `prohow : failed`
`prohow : has_environment`	has_env(y, x)
`prohow : sub_environment_of`	sub_env(y, x)

This interface includes an online editor that translates free text into RDF using the PROHOW vocabulary. This editor parses a user's input and looks for keywords such as "Step" and "Requires" to identify steps and requirements. After parsing the text into an RDF graph, a visualization is provided to the user. If the user is satisfied with the machine interpretation of the instructions, a save button stores the generated know-how as an RDF graph in the knowledge base of the system, minting new URIs for each component of the instructions. Those URIs are dereferenceable, and users can use them to request a human-readable visualization. The same URI can be used by machines to obtain a machine-readable version. This is done by content negotiation, for example by setting the *Accept* header of an HTTP request to *application/rdf+xml*. The graphical interface also supports users during the know-how linking and execution phases. It allows them to create or link steps, methods and requirements and to initiate and update task executions. The user can choose whether to complete all tasks manually, or to let the system automate them whenever possible.

7 Experiments

The objective of this experiment is to support our hypothesis that the majority of (non-programmer) Web users can define certain tasks at a level of abstraction

which is understandable by machines. We evaluated this hypothesis with respect to the three phases of our framework described in Sect. 4: know-how acquisition, know-how linking and execution. This experiment is also meant to demonstrate the application of our approach in a concrete scenario. Human participation in this experiment has been obtained through the crowdsourcing platform Crowdflower.[8] Participants were asked information about their computer skills to exclude answers from computer experts.

7.1 Evaluation of the Know-How Acquisition Phase

During the know-how acquisition phase, our hypothesis is that humans can write semi-structured procedures which can be automatically parsed into an RDF representation. We evaluated this by asking 10 workers to solve the example task described in Sect. 2 through an online survey.[9] Workers submitted their instructions in a text-box in natural language. To improve the quality of their submissions, workers were asked to follow certain rules, such as to clearly divide their instructions into steps, and were offered a bonus compensation for creating high quality instructions. All the original submissions are available online.[10] Of the 10 submissions we received, we rejected 3 of them as non-genuine attempts. Of the remaining submissions, we considered the 5 best solutions for the next part of the experiment.

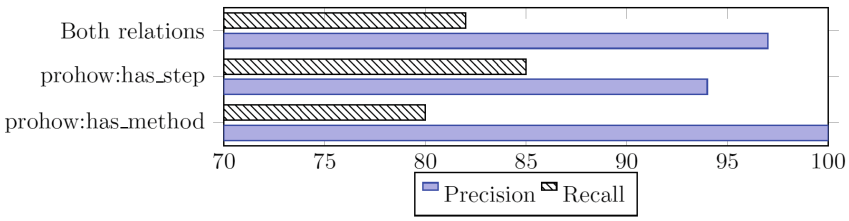

Fig. 2. Precision and recall of the links generated by the workers.

7.2 Evaluation of the Know-How Linking Phase

To judge whether non-programmer humans can correctly link instructions to automatable functions, we have paired each of the 16 steps of the five best sets of instructions with 10 different machine functionalities. All automatable steps have been paired with one or more relevant functionalities, as well as unrelated ones. For each step-functionality pair, we have then asked workers to judge whether a particular functionality (in the survey called *action*), such as

[8] http://www.crowdflower.com/.
[9] http://w3id.org/prohow/r1/survey.
[10] http://w3id.org/prohow/r1/survey_results.

"Remove every occurrence of a particular character from the string of text" is relevant for the execution of a particular step (in the survey called *goal*), such as "Remove < and > from the string of text". Each worker was asked to choose one of the following three answers: (1) "YES, this action can completely achieve the goal", (2) "YES, BUT the action can only achieve part of the goal" or (3) "NO, the action is unrelated with the goal". For some functionalities, workers were also asked to provide information on how the function should be completed, for example by answering the question: "Which is the character to remove?".

For each step-function pair we have asked the judgement of 10 different workers, and then we have chosen the judgement given by the majority of the workers. For all questions, the most common answer was always chosen by more than 50 % of the workers. To judge whether an answer is correct or not, we interpret the first answer as the creation of a `prohow:has_method` link between the step and the functionality; the second answer as the creation of a `prohow:has_step` relation, and the third answer as no relation. We have manually evaluated the links generated by the majority of the workers and the precision and recall of those links is shown in Fig. 2. The result of this evaluation shows that the majority of the workers correctly chose to create a link between a step and a functionality 97 % of the time, discovering 82 % of all possible correct links.

7.3 Demonstration of the Execution Phase

To enable collaborative human-machine execution we have developed the machine functionalities which workers previously created links to. Our system listens to changes to its knowledge base to detect when and how its functionalities are needed. When this happens, the system will execute the functionality and modify the knowledge base accordingly, so as to allow the human user to notice that a task has been accomplished. As a result of this experiment, all the five sets of instructions are now available online,[11] and each of them contains at least one automatable function.

8 Related Work

We frame our work at the intersection of several related areas, which highlight its interdisciplinary nature. The idea of combining human and machine efforts to solve tasks that neither humans or machines alone could solve efficiently is central to the field of Human-Computation [8]. In Human Computation systems, humans typically play a subordinate role, as they have no direct control over the computation, and are not in charge of initiating it. For example, users of the Galaxy Zoo[12] project are asked to detect patterns in sky images to help machines to classify galaxies. However, they have no control over the machine computation that utilizes their contributions. More control is given to users

[11] http://w3id.org/prohow/r1/instructions.
[12] http://www.galaxyzoo.org/.

of Human-Provided Services (HPS) [10]. For example, HPS users can actively define and advertise the services they want to offer, and manage their interactions with other users. In general, Human Computation is better suited to accomplish large and complex tasks, while HPS can better address dynamic tasks with rich user interactions. However, both of them require expert intervention to define task workflows and therefore neither of them can effectively address the goals of individual users. The main objective of our work, instead, is to put humans in control of the computation. In our framework, humans participate to solve tasks that they have defined and that they are directly interested in accomplishing.

In our approach, non-programmer users define workflows by providing natural language instructions. While we limit the analysis of these instructions at the structural level (e.g. steps and requirements), the possibility of extending this analysis by means of Natural Language Processing has been investigated in the literature. For example, an approach has been developed to translate if-this-than-that constructs from natural language into executable programs [9]. Other approaches are more domain specific, such as focusing on cooking recipes [4]. Similarly to Controlled Languages, most of those approaches rely on certain structural or lexical properties of the instructions to extract their meaning.

Several languages have been created to describe processes in different fields, most notably OWL-S [5] in the Semantic Web community. While the majority of these languages focuses on describing automated functionalities, such as Web Services, some languages also include human participation in the computation. For example, the CompFlow [3] ontology allows the definition of workflows that can interleave both human and machine computation. None of these languages, however, is meant to be understandable by generic Web users, and experts are required to define the specific workflows. Moreover, most of these formalisations are too domain specific or logic heavy to conveniently represent human know-how. For example, human tasks are incompatible with OWL-S, since this ontology defines a process as a "specification of the ways a client may interact with a service" [5]. It should be noted, however, that integrations between PROHOW and other languages are possible. For example, the URI of a PROHOW task could point to an OWL-S service that might accomplish the task once invoked.

In our framework, humans and machines interact through the modification of a shared RDF knowledge base. This type of indirect communication resembles Blackboard Systems (BS) [1]. In BS, multiple agents collaborate to compute the solution to a problem by modifying a shared resource (the blackboard). From BS originates another related communication mechanism: Triple Space Computing (TSC) [2]. In the TSC approach, coordination between Semantic Web Services is achieved indirectly by publishing and reading RDF resources on the Web, which are organised into *Triple Spaces*. TSC includes several functionalities, such as the possibility to create, advertise or subscribe to particular Triple Spaces. Although useful in a more generic setting, such functionalities are not currently included in our framework, which can be imagined as having a single Triple Space.

9 Conclusion

In this paper we have addressed the problem of enabling non-programmer humans to specify the type of behaviour they require from machines, as opposed to being constrained by pre-defined functionalities. To solve this problem we have presented the first framework that allows non-programmers to define and execute human-machine workflows—that is, workflows that combine human and machine actions to achieve a common goal.

Human-machine collaboration is achieved by indirect communication through a shared Linked Data knowledge base defined with our PROHOW vocabulary. The logical interpretation associated with this vocabulary makes the knowledge base machine understandable, allowing machines to infer when and how their functionalities are required. At the same time, this knowledge base is made human understandable by a direct visualization of its contents through an intuitive web interface. Using this interface, user-generated instructions in natural language are automatically translated into Linked Data.

Unlike a modelling language, the main objective of our vocabulary is not to describe how humans and machines collaborate, but rather to enable this collaboration in practice. To demonstrate this, we presented an implementation of our framework which we evaluated in a concrete test scenario. The results of this experiment support our hypothesis that non-programmers can specify certain types of instructions at the level of detail required for machine understanding.

References

1. Corkill, D.D.: Blackboard systems. AI Expert **6**(9), 40–47 (1991)
2. Fensel, D., Facca, F.M., Simperl, E., Toma, I.: Triple space computing for semantic web services. In: Semantic Web Services, pp. 219–249 (2011)
3. Luz, N., Pereira, C., Silva, N., Novais, P., Teixeira, A., Oliveira e Silva, M.: An ontology for human-machine computation workflow specification. In: Polycarpou, M., de Carvalho, A.C.P.L.F., Pan, J.-S., Woźniak, M., Quintian, H., Corchado, E. (eds.) HAIS 2014. LNCS, vol. 8480, pp. 49–60. Springer, Heidelberg (2014)
4. Malmaud, J., Wagner, E.J., Chang, N., Murphy, K.: Cooking with semantics. In: Proceedings of the ACL 2014 Workshop on Semantic Parsing, pp. 33–38 (2014)
5. Martin, D., Burstein, M., Hobbs, J., et al.: OWL-S: Semantic markup for web services. W3C Member Submission (2004)
6. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: environment-based coordination for intelligent agents. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 286–293 (2004)
7. Pareti, P., Testu, B., Ichise, R., Klein, E., Barker, A.: Integrating know-how into the linked data cloud. In: Janowicz, K., Schlobach, S., Lambrix, P., Hyvönen, E. (eds.) EKAW 2014. LNCS, vol. 8876, pp. 385–396. Springer, Heidelberg (2014)
8. Quinn, A.J., Bederson, B.B.: Human computation: a survey and taxonomy of a growing field. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1403–1412 (2011)

9. Quirk, C., Mooney, R., Galley, M.: Language to code: learning semantic parsers for if-this-then-that recipes. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL–2015), pp. 878–888 (2015)
10. Schall, D.: Service-Oriented Crowdsourcing: Architecture Protocols and Algorithms, chapter Human-Provided Services, pp. 31–58 (2012)
11. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)