Representation and Execution of Human Know-How on the Web

Paolo Pareti

Doctor of Philosophy
Centre for Intelligent Systems and their Applications
School of Informatics
University of Edinburgh
2017

Abstract

Structured data has been a major component of web resources since the very beginning of the web. Metadata that was originally mostly meant for display purposes gradually expanded to incorporate the semantic content of a page. Until now semantic data on the web has mostly focused on factual knowledge, namely trying to capture "what humans know". This thesis instead focuses on procedural knowledge, or in other words, "how humans do things", and in particular on step-by-step instructions. I will present a semantic framework to capture the meaning of sets of instructions with respect to their potential execution. This framework is based on a logical model which I evaluated in terms of its expressiveness and it compatibility with existing languages. I will show how this type of procedural knowledge can be automatically acquired from human-generated instructions on the web, while at the same time bridging the semantic gap, from unstructured to structured, by mapping these resources into a formal process description language. I will demonstrate how procedural and factual data on the web can be integrated automatically using Linked Data, and how this integration results in an overall richer semantic representation. To validate these claims I have conducted large scale knowledge acquisition and integration experiments on two prominent instructional websites and evaluated the results against a human benchmark. Finally, I will demonstrate how existing web technologies allow for this data to seamlessly enrich existing web resources and to be used on the web without the need for centralisation. I have explored the potential uses of formalised instructions by the implementation and testing of concrete prototypes which enable human users to explore know-how and collaborate with machines in novel ways.

Acknowledgements

I am most thankful to my supervisors, Prof. Ewan Klein, Dr. Adam Barker and Dr. Michael Rovatsos, for constantly guiding and inspiring me. Much of what I learnt, I owe to them. I would also like to thank my examiners, Dr Alasdair J. G. Gray and Prof. Stuart Anderson, for all their work in reviewing this thesis.

I am very grateful to my sponsors, the Scottish Informatics and Computer Science Alliance, the ESSENCE Marie Curie Initial Training Network and Taiger Spain S.L. for supporting me throughout my studies and for providing me with the opportunity to attend conferences, summer schools, and other excellent events.

I am also thankful to Prof. Ryutaro Ichise with whom I collaborated at the National Informatics Institute in Tokyo. Prof. Ryutaro's excellent supervision and support had a significant impact on my PhD. And I am thankful to Benoit Testu, for all the hard work we did together.

And of course, I am especially thankful to my many friends and colleagues, with whom I shared countless adventures and innumerable cups of coffee.

Declaration

Table of Contents

Chapter 1

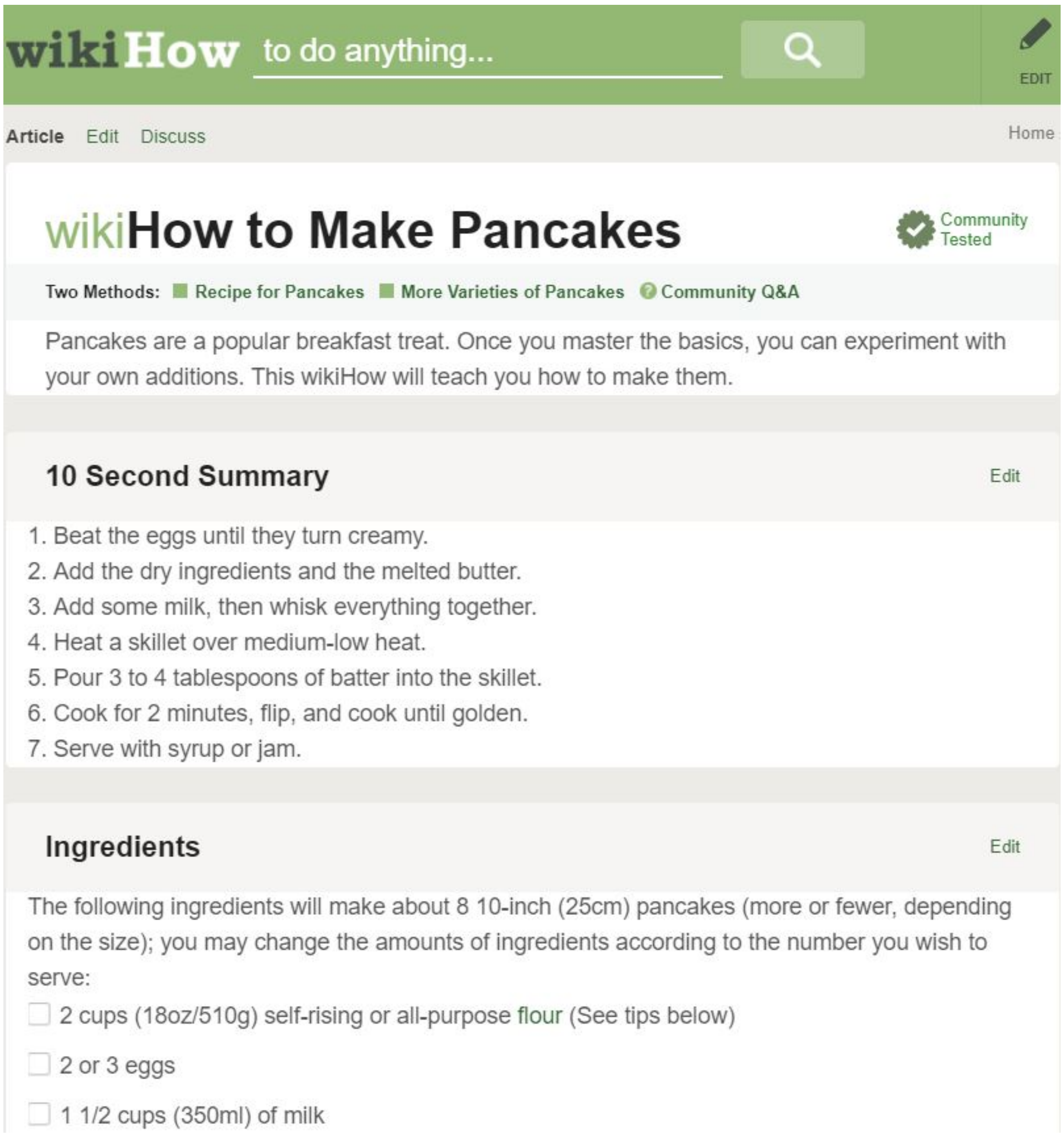
Introduction

Structured data has been a major component of web resources since the very beginning of the web. Metadata that was originally mostly meant for display purposes, instructing Web browsers on how to visualise information, gradually expanded to incorporate application logic for more dynamic and interactive pages. In recent years, metadata in web resources started describing also the semantics of their content. This type of semantic metadata is already being used for a large variety of applications, which range from better indexing and retrieval of web documents,[1] to semantic exploration of web content, such as Google info-boxes, to populating databases of common sense *knowledge*, such as DBpedia (Auer et al. 2007).

Following Bellinger et al. (2004), I use the term 'knowledge' to refer to a *more* understandable version of data. Knowledge can be seen as data that is organised according to some known rules and patterns that allow us to infer new facts or make predictions. For example, knowledge of the recipe for a certain dish typically allows us to infer whether a certain set of ingredients are sufficient to prepare it.

Until now these semantic technologies have mostly focused on factual knowledge, namely trying to capture *what humans know*. However, as noticed by Thimm et al. (2012), there is a different class of knowledge that, despite being common on the web and in our daily life, does not feature as often in this web of data. This class of knowledge, which will be the focus of this thesis, is *know-how*, or *procedural knowledge*, which captures *what humans do, and how*. More specifically, I will use the terms know-how and procedural knowledge interchangeably to refer to knowledge about how humans do things that can be made explicit and communicated through language. Implicit know-how, such as the motor skills required to ride a bicycle, is outside of the

[1] `http://schema.org/` (accessed on 17/10/2017)


wikiHow to do anything...

EDIT

Article Edit Discuss Home

wikiHow to Make Pancakes

Community Tested

Two Methods: Recipe for Pancakes More Varieties of Pancakes Community Q&A

Pancakes are a popular breakfast treat. Once you master the basics, you can experiment with your own additions. This wikiHow will teach you how to make them.

10 Second Summary Edit

1. Beat the eggs until they turn creamy.
2. Add the dry ingredients and the melted butter.
3. Add some milk, then whisk everything together.
4. Heat a skillet over medium-low heat.
5. Pour 3 to 4 tablespoons of batter into the skillet.
6. Cook for 2 minutes, flip, and cook until golden.
7. Serve with syrup or jam.

Ingredients Edit

The following ingredients will make about 8 10-inch (25cm) pancakes (more or fewer, depending on the size); you may change the amounts of ingredients according to the number you wish to serve:

- ☐ 2 cups (18oz/510g) self-rising or all-purpose flour (See tips below)
- ☐ 2 or 3 eggs
- ☐ 1 1/2 cups (350ml) of milk

Figure 1.1: Partial screenshot of a set of instructions on how to make pancakes.[2]

scope of this thesis.

Procedural knowledge is available on the web in many formats, such as text, images and videos. In this thesis I will focus on textual instructions, one of the most common formats used to share procedural knowledge online. Textual instructions can be seen as descriptions in natural language that answer the question of *how* a certain task can be accomplished. A concrete example of the type of instructions that this thesis will focus on is displayed in Figure 1.1. While there is no universal way to represent instructions, a number of common structures can be identified in this example. For example, this

[2]Screenshot taken from `http://www.wikihow.com/Make-Pancakes` on date 16/10/2017.

set of instructions features a title, namely “How to Make Pancakes”, which describes the main goal that the instructions are meant to achieve. Below this title we can notice an ordered list of steps that details the actions that need to be performed and in which order. For example, the set of instructions should begin with the step “Beat the eggs until they turn creamy”. Finally, we can notice a set of requirements that are needed by the set of instructions, such as the ingredient “2 or 3 eggs”.

1.1 Problem Statement

The central **problem** addressed in this thesis is the following: human instructions on the web are not machine-understandable. In other words, these instructions are used as *knowledge* by humans, but are currently only processed as *data* by machines. Recent research efforts targeting human instructions, such as the works by Beetz et al. (2015) and Kiddon et al. (2015), can be seen as evidence of the importance of this problem.

This problem significantly limits the support that machines can offer to users who are interested in this type of knowledge. While a human can perform complex tasks after learning the relevant know-how from a resource containing a set of instructions, a machine given the same resource can typically only perform a number of basic tasks, such as displaying its contents on a web browser. We do not currently expect machines to exhibit more complex behaviours, such as using this knowledge to help us executing the set of instructions or to automatically answer complex questions, such as “what are the most common steps to make pancakes?”.

An important aspect of this problem is the focus on web resources. Machine-understandable know-how should be publishable and shareable on the web in a similar fashion to current instructional web-pages. Even if a system achieves a high level of machine understanding of human instructions, such a system would not completely solve the problem above if its knowledge is isolated, and if there is no strategy in place to allow other systems to access it.

The main **goal** of this thesis is to enable the development of intelligent systems by addressing the problem of human instructions on the web not being machine-understandable. I break down this problem into three main sub-problems, and their related research questions:

Know-How Representation How can we formalise instructional knowledge in a form that is machine-understandable and that can be shared on the web?

Know-How Acquisition How can we acquire a machine-readable formalisation of instructions from existing instructional resources?

Know-How Applications How can we use a machine-readable formalisation of instructions to support novel applications, including applications that support the execution of such instructions?

Without the availability of machine-understandable instructional resources, machines would require highly sophisticated systems to interpret raw instructional resources every time they need to access them. This is an approach that I deem impractical with current technologies and that I will not consider in this thesis.

Since the current representation of instructions on the web is not understandable by machines, I propose a different representation which will remedy this limitation. The *know-how representation* question is concerned with the choice of a suitable knowledge representation formalism to encode instructional knowledge in a way that can be easily accessed on the web and processed by machines.

Having defined a formalism, the *know-how acquisition* question is concerned with methods to populate a knowledge base with the formal representation instructions. In other words, we require methods to generate a machine-understandable version of instructions from available resources, such as existing instructions on the web. Without practical and proven strategies to perform knowledge acquisition, a semantic representation formalism risks being only a theoretical exercise which cannot be applied to the real world.

The *know-how application* question is concerned with assessing the extent to which a given formalism can be considered *more* machine-understandable by considering the type of applications that it enables which were not previously possible. Although no knowledge representation formalism can be optimal in all scenarios, an indication of its quality comes from its re-usability in several different applications. The primary objective of knowledge representation should be to represent a domain, and not just to serve a specific function within a system. A good representation should therefore capture useful semantic properties of a domain which can be used in different applications.

It is important to address these three questions at the same time, since the complexity of the know-how acquisition and the know-how applications questions depend on the choice of the know-how representation formalism. In particular, it depends on the semantic richness of this representation. While there is no universal metric of semantic richness, I will consider the number-of-features approach (Pexman et al. 2008), which I demonstrated to be effective when applied to distributed knowledge bases (Pareti et al. 2015). Following this measure, a knowledge representation formalism is said to be semantically richer the more facts its statements allow us to infer. For example, the representation of someone as a "student" instead of a "person" can be said to be semantically richer, as more assumptions can be made about a "student" than about a "person".

With this definition in mind, it is usually the case that the more semantically rich a representation is, the easier it is to support intelligent applications, but the harder it is to acquire it. This difficulty in acquisition is usually due to the difficulty of ensuring the correctness of detailed information and all the possible facts that it can entail. On the other hand, semantically poor knowledge bases, while easy to generate, tend to be vague and are not expressive enough to have many practical uses. For this reason, it can be said that know-how acquisition and know-how applications result in conflicting requirements in the choice of a representation.

The importance of addressing these three questions at the same time is supported by previous work in the domain of factual knowledge. In the paper by Auer et al. (2007) that presented DBpedia to the research community, the authors address a similar set of problems. Auer et al. define a formalisation, the DBpedia ontology, and present an extraction framework capable of acquiring knowledge in this formalisation from Wikipedia. At the same time, they provide a set of interfaces and applications that demonstrate possible uses of this dataset.

I will now summarise the current state of the art with respect to these research questions and identify the gaps in the literature that need to be filled to solve the central problem discussed in this thesis, namely the low level of machine understanding of instructions on the web.

1.2 Overview of Related Literature

Know-How Representation In order to improve the level of machine understanding of instructions on the web, the first question that needs to be addressed is how to represent this knowledge on the web. This question can benefit from the significant body of work on knowledge representation. In particular, a number of formal languages and technologies have been developed within the Semantic Web field to represent and share knowledge on the web. I will present a detailed review of these approaches in Section

2.1, as they represent the backbone of a significant portion of this thesis' work.

While Semantic Web technologies can be used to represent semantic data on the web, they do not define what its semantics should be. A large number of formalisms to define procedural knowledge have been developed and I will review the most relevant ones in Section 2.2. The vast majority of these formalisms, however, have been designed to represent specialised types of procedures such as planning languages or business workflows. These formalisations of well-defined domains are in stark contrast with the open-ended nature of human know-how, which relies on implicit common sense knowledge that cannot be defined in exact terms. As a result of this, these formalisms do not lend themselves well to the domain of human instructions. Existing attempts at formalising human instructions have been carried out in a number of projects. These formalisation attempts, however, were focused on supporting a specific application and were not meant to be reusable and shareable on the web. A formalisation of human instructions as reusable knowledge, therefore, is largely absent from the literature.

Know-How Acquisition Several approaches to obtaining instructional knowledge will be presented in Section 2.3. A number of these approaches can be used to extract know-how from web instructions, and therefore can be seen as possible answers to the know-how acquisition question. These approaches, however, suffer from limitations such as domain specificity and significant loss of precision and incompleteness of the resulting formalisation. A solid and domain independent method for formalising instructional knowledge is still missing from the literature.

Know-How Applications Several applications of instructional knowledge will be reviewed in Section 2.4. These applications demonstrate some of the possible uses of this type of knowledge. However, they make use of *ad hoc* formalisations which are designed to achieve a single task and which have not been tested in multiple scenarios. A demonstration of the possibility of using a single know-how representation of instructions to support multiple applications is still missing from the literature.

1.3 Contributions

As its main contribution, this thesis describes the first method to address the problem to represent machine-understandable instructional resources on the web by *simulta-*

neously addressing all three of its sub-problems, namely know-how representation, know-how acquisition and know-how applications. More precisely, I start by defining a knowledge representation formalism to capture human instructions, thus addressing the know-how representation problem. I then proceed to address the know-how acquisition and know-how applications sub-problems with respect to this formalism.

In this thesis I aim to achieve both theoretical and practical goals. I do so by interleaving formal and experimental work to develop theoretical frameworks which are practical, and concrete systems which are well founded. For this reason, the research outputs of this thesis can be broken down into a number of theoretical and experimental contributions. Contributions 1, 2, 3 and 8 represent the main theoretical contributions. Contributions 4, 5, 6 and 7 represent tools and resources that I have developed as proofs of concepts.

The central piece of work on top of which the other components of this thesis are based are the following:

Contribution 1 (PROHOW: A Concrete Know-How Formalisation)**.** I define PROHOW, a knowledge representation formalism for instructional knowledge at a level of abstraction which is understandable by both humans and machines. On the human side, this model directly maps to concepts that are commonly found in human instructions, such as “steps” and “requirements”. On the machine side, the model provides (1) a logical interpretation capable of supporting reasoning and inferring new facts, and (2) a mapping to an automated planning language to reason about the execution of instructions.

The PROHOW formalism provides a solution to the know-how representation problem. Given this formalism, I thus provide a solution to the know-how acquisition and know-how applications problems. In order to solve the know-how acquisition problem, I have developed the following two methods:

Contribution 2 (Know-How Acquisition Method)**.** I provide a know-how acquisition method that automatically extracts a complete and correct PROHOW representation of a common class of online instructional resources, namely *semi-structured* resources. I validate this method with a large-scale know-how acquisition experiment on two major instructional websites.

Contribution 3 (Know-How Interlinking Method)**.** I provide an interlinking method that enriches PROHOW know-how with a semantic context by integrating it with related knowledge. This method links the PROHOW representation of different instruction sets with each other, and with DBpedia. I have tested the quality of the links generated by this method and I have shown how they outperform an existing human-generated baseline.

The combination of the know-how acquisition and interlinking methods results in an automatic approach to generate the PROHOW representation of instructions from existing web documents. To follow best practice for publishing data on the web, I extended this approach into a method for publishing know-how as 5-star Open Data.[3]

The previous pieces of work resulted in the formalisation of a large amount of real-world data that I made available on the web for further reuse.

Contribution 4 (HOWDB: A Large-Scale Know-How Dataset)**.** I have published a dataset of over $200,000$ sets of instructions formalised using the PROHOW model, and a dataset containing over $700,000$ sets of instructions formalised in 15 different languages. Publishing this data improves experiment reproducibility and, in line with the main goal of this thesis, provides a tool to support further research on this topic.

Finally, I provide a solution to the know-how applications problem by demonstrating two different uses of the PROHOW formalisation of instructions. More specifically, I demonstrate how the proposed formalisation can support web users in exploring instructional knowledge and in following a set of instructions.

Contribution 5 (HowLinks: A Know-How Exploration Tool)**.** I have co-developed HowLinks, an exploration tool based on PROHOW formalisations. This tool allows users to explore knowledge related to instructions in novel ways which were not possible in their original form. For example, the HowLinks tool can be used to provide an integrated view of related sets of instructions enriched with DBpedia information.

Contribution 6 (Machine-Assisted Execution of Instructions)**.** I have developed a system that supports a user in the execution of a set of instructions by keeping track of the progress of the execution and automating certain steps of the instructions when possible. I have tested this system with human participants to gain insights into whether users with no programming experience can correctly use this system to create sets of instructions and correctly link them to automatable machine functionalities.

In the context of machine-understandable web resources, I have also investigated whether it is possible to develop similar applications in a decentralised scenario. In a

[3]`http://5stardata.info/en/` (accessed on 17/10/2017)

decentralised scenario, I do not assume the availability of a central dataset collecting the know-how of multiple sets of instructions. Instead, I consider PROHOW formalisations of instructions to be decentralised similarly to how instructions on the web are currently decentralised across a large number of different web pages.

Contribution 7 (KnowHow4j: A Decentralised Know-How Exploration Tool)**.** I have developed KnowHow4j, an exploration tool that allows users to generate and explore decentralised PROHOW resources. This tool is based on the dynamic discovery of web data at run time. In other words, web data is discovered, retrieved and integrated dynamically when it is needed by a user. This tool can also be used by users to generate HTML instructional pages which contain an embedded PROHOW representation of the instructions.

Contribution 8 (Decentralised Human-Machine Collaboration Framework)**.** I have propose a conceptual framework that allows multiple agents, both humans and machines, to collaborate in the execution of a set of instructions in a decentralised fashion.

1.4 Overview of the Thesis

This thesis is organised as follows:

Chapter 2: Related Work I will start this thesis with a review of the most relevant literature to the problem at hand. This review is organised around four themes. The first one is about the Semantic Web technology stack to publish semantic data on the web. The second one is about previous work on formalisations of procedural knowledge. Together, the first and second themes describe the state of the art with respect to the know-how representation problem. The third theme is about the existing literature on acquiring instructional knowledge from users and from online instructions. As such, it describes the state of the art with respect to the the know-how acquisition problem. The last theme addresses the know-how applications problem by describing prior work on applications of formalised instructions.

Chapter 3: Methodology In this chapter I present the different methodologies that will be used to evaluate my work. The contributions of this thesis, in fact, span across different fields, and no single unified methodology exists to evaluate them.

Chapter 4: Know-How Representation In this chapter I present a mathematical model and a concrete formalisation of the human instructions domain. This chapter presents Contribution 1.

Chapter 5: Know-How Acquisition In this chapter I present a method for acquiring instructional knowledge from existing resources and I describe its experimental evaluation using real-world data. This chapter presents Contribution 2.

Chapter 6: Know-How Interlinking In this chapter I present a method for integrating know-how with related knowledge. I apply this method to the data obtained in the previous chapter and evaluate the results against a human benchmark. This chapter presents Contributions 3 and 4.

Chapter 7: Know-How Applications In this chapter I present two different applications of the proposed know-how formalisms. Each of these applications is discussed in a centralised and in a decentralised scenario. This chapter presents Contributions 5, 7, 6 and 8.

Chapter 8: Conclusion In this last chapter I summarise the main findings of this thesis. I discuss the extent to which this work has solved the main problem addressed by this thesis and present opportunities for further research.

Related Publications

The work presented in this thesis resulted in a number of academic publications.

- Pareti, P., Klein, E., and Barker, A. (2016). Linking Data, Services and Human Know-How. In *The Semantic Web. Latest Advances and New Domains*, ESWC 2016, pages 505–520. Springer International Publishing

  Conference paper presented at the 2016 Extended Semantic Web Conference (ESWC). This publication refers to Contribution 6 presented in this thesis of which I am the main author.

- Pareti, P., Testu, B., Ichise, R., Klein, E., and Barker, A. (2014b). Integrating Know-How into the Linked Data Cloud. In *Knowledge Engineering and Knowledge Management*, volume 8876 of *Lecture Notes in Computer Science*, pages 385–396. Springer International Publishing

  Conference paper presented at the 19th International Conference on Knowledge Engineering and Knowledge Management (EKAW). This publication refers to Contributions 3 and 5 presented in this thesis of which I am the main author, except for the development of the HowLinks applications, which I co-developed with Benoit Testu.

- Pareti, P. (2016). Distributed Linked Data as a Framework for Human-Machine Collaboration. In Hartig, O., Sequeda, J., and Hogan, A., editors, *Proceedings of the 7th International Workshop on Consuming Linked Data (COLD)*, number 1666 in CEUR Workshop Proceedings, Aachen

  Workshop paper presented at the 7th International Workshop on Consuming Linked Data, co-located with the 15th International Semantic Web Conference (ISWC). This publication refers to Contribution 8 presented in this thesis.

- Pareti, P., Klein, E., and Barker, A. (2014a). A Semantic Web of Know-how: Linked Data for Community-centric Tasks. In *Proceedings of the 23rd International Conference on World Wide Web Companion*, pages 1011–1016

  Workshop paper presented at the 6th International Workshop on Web Intelligence & Communities, co-located with the 23rd International Conference on the World Wide Web (WWW). This publication refers to Contribution 5 presented in this thesis of which I am the main author.

Chapter 2

Related Work

In this chapter I will describe the main research areas that relate to the representation of human instructional knowledge on the web. Sections 2.1 and 2.2 describe respectively, the state of the art in representing knowledge on the web, and on formalising procedural knowledge. These sections relate to the *know-how acquisition* research problem. In Section 2.3 I will presented existing research on acquiring instructional knowledge from web resources, and from users with no programming experience. In Section 2.4, I will review specific applications of formalised instructional knowledge. These last two sections relate, respectively, to the *know-how acquisition* and *know-how applications* research problems.

2.1 Representing Data and Knowledge on the Web

The web has become a place where data is published, discovered and accessed. Publishing data on the web is particularly important for applications that use distributed or decentralised data sources, or to allow datasets to be re-used. This section reviews the main related research areas, namely Linked Data, Ontologies and the Semantic Web.

2.1.1 The Uniform Resource Identifier (URI)

When data is shared between different organisations, a critical problem is to make sure that the same entities and concepts encoded in a message at the source can be unambiguously interpreted by the receiver. To overcome this problem Uniform Resource Identifiers, or URIs, can be used. A URI can be used to unambiguously identify concepts and entities over the internet. When used correctly, URIs guarantee that two identifiers are the same only when they were meant to refer to the same entity or concept. In its most generic formulation[1], the syntax of a URI is made of the following components:

```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```

The `scheme` component specifies one of a limited set of canonical ways to identify a resource. Some valid scheme examples are `http`, `ftp`, `file` and `mailto`. While certain schemes such as `http` define resources at a web scope, others such as `file` can define resources also at a local scope. For example, the same URI `file:///doc.txt` can be used by two different file-systems to refer to two different resources. The use of URIs in data-sharing scenarios is almost exclusively concerned with web-scope identifiers. In such context, the term URI has practically become a synonym of a web-scope identifier, typically with scheme `http`. The use of the `user`, `password` and `port` components is also very rare, and therefore, in practice, URIs tend to be traditional URLs, of which URIs are a generalisation. In this thesis, the term URI will be used in a similar way to refer to URIs which are also web-scale identifiers pointing to potentially web-retrievable resources, such as those using the `http`, `https` and `ftp` schemes.

An extension to URIs which is worth mentioning is the Internationalized Resource Identifier, or IRI which allows non-latin characters to be used in the identifier. While conceptually equivalent, IRIs adoption is still limited and many tools still limit their support to URIs. This is why the term URI has been preferred in this thesis, although the results presented can be equally applied to IRIs.

The main purpose of URIs is to identify specific resources. Special kinds of URIs, however, can be used to encode more information. For example, we can observe that URLs are a type of URI. The main function of a URL is to locate a resource on the web, most commonly through the HTTP protocol. It is therefore possible to create a URI which also acts as a URLs. If the HTTP protocol is used in the definition of such URIs, they can be called HTTP URIs. In other words, an HTTP URI can be used both to identify a concept and to retrieve a certain resource.

When a new concept is created, a URI to refer to it might not exist. It is therefore necessary to create, or *mint*, a new URI to refer to that concept. In order to mint a new URI, it is important to choose a good strategy that guarantees its uniqueness. While it is acceptable that multiple URIs might refer to the same concept, the same URI should never refer to different concepts, as this would most likely lead to semantic

[1]`https://tools.ietf.org/html/rfc3986` (accessed on 17/10/2017)

inconsistencies.

2.1.2 Resource Description Framework (RDF)

The URI identifiers described in the previous section are the building blocks of the Resource Description Framework (RDF) data model (Raimond and Schreiber 2014). RDF incorporates URIs, useful to identify entities, into a complete graph data structure. An RDF dataset can be defined as a set of *graphs*. Graphs, which are identified by a URI, can be used to partition data into different sets. This can be useful to organise data, to attach metadata information, such as provenance, to a specific set of data, or to improve query performance by limiting the scope of queries to specific graphs.

Each graph can then be defined as a set of *triples* $< S, P, O >$, each one consisting of a subject S, a predicate P and an object O. Conceptually, a triple is a statement that a certain relation P exists between a subject entity S and an object entity O. For this reason, RDF graphs can be used to represent labelled directed graphs.

While a shared data model is crucial to ensure the interoperability between multiple datasets, the exact format in which data is serialised is not. For this reason, the RDF data model can be encoded in a number of serialisations. The Terse RDF Triple Language, or Turtle, is an RDF textual serialisation aimed at improving human-readability of RDF data (Carothers and Prud'hommeaux 2014). Like in most other serialisations, *namespace prefixes* are used to shorten the character length of individual URIs by simplifying the recurrent base prefix, or *namespace*. For example, the following line of text assigns the namespace `http://foo.org/` to the prefix `foo`:

```
@prefix foo: <http://foo.org/> .
```

Given this prefix, the triple asserting that the entity of the city of Edinburgh (identified by URI `http://foo.org/E01`) is located in Scotland (identified by URI `http://foo.org/Scotland`), given URI `http://foo.org/located_in` denoting the "located in" relation, can be written in more readable form as follows:

```
@prefix foo: <http://foo.org/> .
foo:E01 foo:located_in foo:Scotland .
```

Table 2.1 contains a list of namespaces that will be used throughout this thesis.

Prefix	Namespace
`prohow:`	`http://w3id.org/prohow#`
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`
`skos:`	`http://www.w3.org/2008/05/skos#`
`foaf:`	`http://xmlns.com/foaf/0.1/`
`dbr:`	`http://dbpedia.org/resource/`
`dbo:`	`http://dbpedia.org/ontology/`
`pto:`	`http://www.productontology.org/id/`
`qudt:`	`http://qudt.org/schema/qudt/`
`unit:`	`http://qudt.org/vocab/unit/`
`:`	`http://example.org/`

Table 2.1: The RDF namespaces used in this thesis.

2.1.3 The SPARQL Protocol and RDF Query Language (SPARQL)

The SPARQL Protocol and RDF Query Language, or SPARQL, (Seaborne and Harris 2013) is the query language to query RDF data recommended by the World Wide Web Consortium (W3C).[2] SPARQL queries operate over RDF data and require a SPARQL software implementation. Most RDF-focused databases, or *triplestores*, such as Virtuoso[3], incorporate a SPARQL implementation. Triplestores often allow SPARQL queries to be executed through a web service accessible over HTTP called a SPARQL *endpoint*.

Like other graph query languages, such as Cypher,[4] SPARQL queries are centred around the concept of *graph pattern matching*. In its simplest form, a SPARQL graph pattern can be seen as a set of potentially incomplete RDF triples, where the subject, predicate or object of triples can be substituted with variables. In the SPARQL syntax, variables are denoted by a prefixed question marks. A result of a simple graph matching query is a mapping $V \mapsto T$, called *variable binding*, between the set of variables V declared in the query and the set of RDF *terms* T, which is the set contains all possible URIs, literals and blank nodes. A correct answer to the query is a mapping such that the graph obtained by substituting each variable in V with the corresponding mapped RDF term in T is a sub-graph of the queried RDF dataset. A complete answer set is

[2]`https://www.w3.org/` (accessed on 17/10/2017)
[3]`https://virtuoso.openlinksw.com/` (accessed on 17/10/2017)
[4]`http://neo4j.com/docs/stable/cypher-introduction.html` (accessed on 17/10/2017)

the set of all possible correct answers, or in other words, the set of all valid variable bindings.

The queries discussed so far are of the SELECT type, which return a variable binding as a result. Other possible SPARQL queries are ASK queries, which return a boolean value to indicate whether a query has at least one solution and CONSTRUCT queries, which return a valid RDF dataset as a result. A specification also exists for SPARQL UPDATE (Passant et al. 2013) queries, which are used not to retrieve answers from an RDF dataset, but to modify it. UPDATE queries can be used, among others, to create or delete triples and graphs.

2.1.4 Vocabularies and Ontologies

While URIs allow the precise identification of entities across multiple databases, they do not contain the meaning of such entities. In other words, the occurrence of the same URI across two datasets implies that both datasets are referring to the same entity. However it might not be known what this entity is.

It becomes therefore important to share not only the identifiers to the same entities, but also their semantic interpretation. For example, the semantic interpretation of the URI `http://foo.org/located_in` should be relation between a city and the country where it can be found.

This semantic interpretation can be described in multiple forms. Typically, one such form is an informal human-understandable description in natural language. A collection of terms described in natural language can be called a *vocabulary*. The main purpose of a vocabulary is to attach a standard semantic definition to a collection of terms. It is also possible to describe the meaning of a term in a more formal way using schema information and logical axioms.

Schema information typically describes the role played by a term within the structure of the data by defining the following information:

- The URIs that denote *classes*, such as the class `foo:city`, denoting the set of things which are cities.
- The URIs that denote *properties*, such as the property `foo:located_in`, denoting a particular relation between a city and a country.
- Hierarchical information about *sub-classes* and *sub-properties*. For example, the class `foo:city` can be a sub-class of the more generic class of a `foo:`

`settlement`.

- The *domain* and *range* of properties. For example, it is possible to state that the subject of the `foo:located_in` property is an entity belonging to class `foo:city`, while its object must belong to class `foo:country`.

Schema definitions typically describe only very basic hierarchical properties. If more expressiveness is needed, logical axioms can be used to formally constrain the meaning of a term. The most common logical formalism used for this purpose is Description Logic (DL), which is described in more detail by Baader et al. (2008). DL statements allow more expressiveness in the definition of a term compared to typical schema information. For example, two classes might be declared as mutually disjoint, or cardinality constraints can be imposed on properties (e.g. a city must be located in at most one country).

The formal definition of terms for the purpose of sharing their semantic interpretation is related to the concept of an *ontology*. The word ontology, originally imported from philosophy, has received numerous definitions within the computer science domain (Giaretta and Guarino 1995). Two different senses of the world ontology have become the most prominent (Chandrasekaran et al. 1999). The first sense refers to the semantic interpretation of a vocabulary of terms, also called *T-Box* from the word “terminology”. According to this sense, an ontology can be defined as an explicit representation of a conceptualisation (Uschold and Gruninger 1996). Often the word “shared” is added to the definition as the typical purpose of ontologies is to enable communication between different agents or systems.

The second sense refers to a set of facts, also called *A-Box* from the word “assertions”, that describe a particular domain using a certain vocabulary. In this work, the terms *ontology* and *vocabulary* will be used to refer to the T-Box sense. The terms *machine-understandable knowledge* and *knowledge base*, instead, will refer to the A-Box sense.

There is no universal definition of the terms *schema* and an *ontology* and these terms are sometime used in a similar way. Both schemas and ontologies allow inferences to be made and both of them use terms which are not completely formally defined. In this thesis, the word *vocabulary* will be used to refer to a collection of terms and their related semantic interpretation, whether it is formally or informally defined. The words schema will refer to a knowledge representation formalism which informally describes a collection of terms and formally describes their hierarchical structure. The word ontology will be used instead to refer to knowledge representation formalism that extends a schema with a formal description of non-hierarchical logical axioms.

2.1.4.1 Vocabularies and Ontologies Definition Languages

Shared vocabularies and ontologies allow people and machines to reach a better understanding the intended meaning of URI identifiers. However, the definition of a vocabulary or an ontology itself presupposes a shared understanding of ontological relations. For example, a common relation between two classes is the *sub-class* relation. A vocabulary might need to specify that class *A* is a sub-class of class *B*. While the URIs used for the entities *A* and *B* could be created locally for the vocabulary, the URI referring to the sub-class relation needs to be globally understandable.

To make ontological relations globally understandable, a number of common standards exist. These standards can be seen as having different levels of expressiveness. One of the most basic of such standards is the RDF Schema, or RDFS, (Brickley and Guha 2014). RDFS is a collection of RDF terms that can be used to define which URIs are to be considered classes (`rdfs:Class`), and which are to be considered properties (`rdf:Property`). Classes and properties can be structured hierarchically using the sub-class (`rdfs:subClassOf`) and sub-property (`rdfs:subPropertyOf`) properties. Properties can also define their domain (`rdfs:domain`) and range (`rdfs:range`). Another common RDFS term is `rdfs:label`, which is used to provide a human readable label to URIs.

RDFS statements can be used to infer new facts. In the standard RDFS entailment regime,[5] the domain and range of a property *r* can be used to infer the class or classes of the subjects and the objects connected with relation *r*. The sub-class and sub-property relations can be used to infer more generic classes and relations whenever the more specific ones are used.

While RDFS defines schemas, the Web Ontology Language (OWL) is one of the most prominent languages to define ontologies (Krötzsch et al. 2012). OWL describes axioms in DL and it represents them using the RDF data model.

[5]`https://www.w3.org/TR/rdf11-mt/#rdfs-entailment` (accessed on 17/10/2017)

2.1.5 Linked Data

The semantic interpretation of an RDF dataset is to a large extent dependent on resources it is linked to, such as other datasets, or the vocabularies and ontologies that it adopted. The discovery of such resources is therefore important in order to reach a correct semantic interpretation of a dataset. Linked Data is a method for publishing data containing URIs, such RDF data, in order to allow the discovery of other related datasets. The basic principle of Linked Data is to identify concepts with HTTP URIs which can be used to locate other Linked Data resources which describe these concepts. The act of retrieving the resource located by an HTTP URI is called URI *dereferencing*. In principle, this process can be repeated after each discovered resource to discover more and more resources. In practice, however, there is no guarantee that HTTP URIs will lead to the discovery of more Linked Data resources.

While Linked Data resources can be explored automatically by machines, they are not human understandable. Human and machine-understandable resources can coexist at the same web locations, i.e. by retrieving the same URL, using technologies such as RDFa or URI dereferencing. RDFa is a standard to embed RDF resources into the metadata of an HTML document. This metadata is not visible when the page is displayed in a web browsers. As a result of this, a human accessing the page would access a human readable document. A machine instead, could parse the HTML page to automatically extract its RDF contents.

A similar result can be achieved by *content negotiation*. Content negotiation is a feature of the HTTP protocol that allows to request a web resource in a specific format by setting the *Accept* header of an HTTP request to the desired format, for example `application/rdf+xml` for a machine readable document, or `text/html` for a human readable one.

2.2 Procedural Knowledge Representation

Procedural knowledge is a common type of knowledge and it can be encountered in in different forms and in many domains. For example, business workflows, planning problems and human instructions can all be considered types of procedural knowledge. I will now review the main types of existing procedural formalisms, and how they relate to human know-how.

2.2.1 Automated Planning languages

One of the most mature fields using process representations is the field of Automated Planning (AP). Within this field, actions are typically described in terms of their inputs and outputs, and their preconditions and effects. The objects and conditions manipulated by actions are usually represented by certain variables which effectively describe a particular state of the world. When an action is allowed in a certain state, its execution can be interpreted as the transition from that state to a different one. The goal of AP algorithms can then be seen as the task of identifying a suitable sequence of actions, also known as a *plan*, to reach a certain goal state from a certain starting state. One of the main goals of AP is the creation of algorithm that can effectively search the space of possible plans to find (or approximate) an optimal plan, under some definition of optimality.

Typical planning description languages like the Planning Domain Definition Language (PDDL) (Mcdermott et al. 1998) adhere to a generic formulation of planning which closely resembles logical proofs. The goal of planning can then be seen as trying to prove a certain goal possible given a set of assumptions. In this case, both the goal and the assumptions can be seen as logical statements. The actions that are allowed in the planning domain can be seen as inference rules, which allow certain new facts (post-conditions) to be proved if certain conditions are met. If found, a proof of this kind can be seen as a plan.

A different approach to planning comes from Hierarchical Task Networks (HTN) (Erol et al. 1994a). The central concept of HTNs is the notion of task decomposition. HTN tasks can be accomplished by accomplishing a set of sub-tasks, thus generating a hierarchical structure which represents the process of achieving a goal at different levels of abstraction. Typically, HTN languages define the concept of *primitive tasks*, namely tasks which can be achieved directly without being further decomposed into simpler tasks. Given a goal task to achieve, the process of planning using HTNs can then be seen as the process of decomposing the goal into a set of sub-tasks such that all of them are primitive tasks. The execution of all such primitive tasks, optionally subject to a particular parametrisation and ordering discovered during planning, will then result in the completion of the goal.

Human instructions are inherently decentralised, and when formalising them, special care should be paid at easing the integration of different sets of instructions. AP process representations, on the other hand, are not designed to be decentralised, and this can cause problems in integrating multiple AP processes coming from different sources. For example, multiple PDDL planning domain descriptions cannot be integrated automatically.

Given the strong emphasis on automation, AP tends to adopt “low-level” process representations. These formalisms can precisely and effectively describe limited domains, such as the movement of boxes in a warehouse, allowing for complex inferences to be made. In “classical planning”, full observability of the world and deterministic actions are usually assumed. This is often in stark contrast to human-generated plans, which are characterised by more underspecified and less predictable actions. Even when AP algorithms deal with uncertainty, such as in “probabilistic planning”, they still rely on detailed descriptions of the world, such as the level of uncertainty associated with the actions.

In this section I have reviewed two foundational approaches of AP which strongly related with the work of my thesis. I have not covered other aspects of AP, such as the specialisation of planning algorithms to more specialised domains, or approaches to improve the efficiency of automated planners. For a more comprehensive survey of this field, see Geffner and Bonet (2013).

2.2.2 Standards for Process Representations

With the objective of improving the interoperability between different process representation, several academic and industrial institution collaborated for the creation of the Process Interchange Format (PIF) ontology (Lee et al. 1998). This ontology is designed to act as an interlingua between different domain specific process representations by capturing the most important and co-occurring concepts. The question of which and how many important concepts to include in the core PIF ontology does not have an exact answer, and the authors of PIF aimed to achieve the best trade-off between simplicity and expressiveness for the most common use-cases. The PIF ontology was later incorporated into the Process Specification Language (PSL) (Grüninger and Menzel 2003), a language more focussed on business and manufacturing use-cases.[6]

PIF does not provide the right level of abstraction needed in this project to model generic human know-how. While some concepts in PIF are not required in a human know-how domain, thus creating redundancy, others do not have an explicit representation, such as the concept of a set of instructions including different “methods”. PSL

[6]`https://www.iso.org/standard/35431.html` (accessed on 17/10/2017)

suffers from similar limitations in modelling human know-how. Moreover, PSL was not designed to be distributed, and dividing a PSL process into several parts might cause them to be locally inconsistent.

Another approach to standardisation comes from the Schema.org[7] initiative. This initiative, backed by some of the major search engine operators, aims to standardise metadata, or data markup, for web resources. To do so, Schema.org provides identifiers for a large number of common classes and relations, such as *Person* and *City*. Since August 2017 Schema.org has introduced a representation of step-by-step instructions with the *HowTo* vocabulary. This vocabulary is a generalisation of a previous *Recipes* vocabulary which is focussed on instructions from the cooking domain. One of the two main components of this vocabulary is an ordered lists of elements which can either be steps, called *HowToStep*, or further *HowToSection* elements. An *HowToStep* element is made of a number of directions and/or tips called, respectively *HowToDirection* and *HowToTip*. The other component of the vocabulary allows to define requirements, called *HowToItem*, which can be specialised in either consumable (*HowToSupply*) or non consumable (*HowToTool*) objects.

The Schema.org HowTo vocabulary provides an efficient representation of simple linear sequences of steps. However, it lacks sufficient expressiveness to model more complex sets of instructions, such as those available on the WikiHow website. For example, this vocabulary cannot, to date, be used to represent alternatives ways, or methods, to achieve the same result. Defining a partial ordering between steps is also not currently possible, as a notion of dependency or temporal constraints between steps is not available on Schema.org yet. While simplicity and ease to use is one of the main strength of the Schema.org vocabulary, this comes at the cost of a lack of precise semantics. For example, it is not explicitly asserted whether steps should be executed sequentially in the same order as they are listed. Similarly, in case multiple groups of steps (*HowToSection*) are present, it remains undefined whether such groups of steps can be performed independently, or on whether the ordering in which they appear in the document should be taken into consideration.

A related project coming from the Semantic Web field has developed the Activity Streams specification (Snell and Prodromou 2017). This specification aims to standardise the formalisation of human activities using Linked Data and it is especially focused on capturing social actions of web users. The Activity Stream specification aims to be simple and it is centred around the concepts of "actors", "activities" and "ob-

[7]`http://schema.org/` (accessed on 17/10/2017)

jects". Intuitively, actors are the entities capable of performing activities that involve certain objects. This specification is not concerned with the notion of *how* complex activities are performed, and it lacks, for example, the notion of how an activity can be decomposed into simpler sub-activities.

2.2.3 Workflows and Web Services

Web Services can be thought of as self contained applications that can be accessed thought internet interfaces. The description of a web service typically describes *what* a certain functionality achieves, but not *how* it is achieved. For example, the Web Services Description Language, or WSDL (Ryman et al. 2007), can be used to define the inputs and outputs of a service.

An important feature of Web Services is the ability to combine multiple services together into workflows. For example, the outputs of a service can be used as inputs for another one. Several languages have been created to represent workflows and their composition (Alonso et al. 2004). In the Semantic Web field, the most prominent one is OWL-S (Martin et al. 2007). OWL-S defines Web Services which can be combined into workflows using *control constructs*, similar to workflow patterns (Van Der Aalst et al. 2003).

Although these type of workflows represent procedural knowledge, they are not suitable to represent human instructions. In OWL-S for example, *processes*, which are the basic building blocks of workflows, are defined as interactions between a client and a service; definition which is arguably not applicable to human instructions. For example, if we state that a task *t*: "fix a broken tire" is an OWL-S process, following the axioms of the OWL-S ontology we must also commit to the fact that *t* is also of type *Service*, of type *ServiceModel*, and exactly one of the the following three types: *AtomicProcess*, *SimpleProcess* and *CompositeProcess*. Each of those classes carries a specific meaning which comes both from an informal definition of the class and from logical axioms. For these reasons, reusing the OWL-S definition of a process in a domain different from web-services, such as the domain of human instructions, can not only lead to misunderstandings, but it can also lead to unwanted inferences and logical inconsistencies when axioms are violated.

Although certain aspects of human instructions, such as step ordering, can be interpreted as workflows, existing workflow languages do not provide a practical way of formalising such instructions. Workflow languages might lack an explicit representation of common concepts like “steps” and “methods”, or might include complex operators that do not have a counterpart in human instructions.

Typically, the basic building blocks of a workflow are assumed to be automated functions. A number of workflow languages, however, consider both humans and machines in their execution. CompFlow, presented by Luz et al. (2014), is an example of this. CompFlow is a workflow definition language similar to OWL-S, but with the main difference of considering both human and machine agents. Another example is CrowdLang, presented by Minder and Bernstein (2012). CrowdLang is a language to model *Social Computation* workflows, and it supports several common Social Computation patterns, such as contests, divide-and-conquer and group decisions. Both CompFlow and CrowdLang, however, are sophisticated workflow languages that require expert knowledge and programming skills to be defined. Their level of complexity is not compatible with common human instructions.

2.3 Know-How Acquisition

Human know-how can be a valuable resource in many applications and several approaches have been developed to extract aspects of it. The majority of the existing approaches focus on obtaining such know-how from human instructions, or by interacting with humans directly. However other less common resources, such as Twitter[8] tweets, have also been considered.

2.3.1 Know-How Acquisition from Instructions

Most approaches to acquire know-how from instructional documents come from the area of Natural Language Processing (NLP). An early application of NLP to formalise wikiHow instructions comes from Addis and Borrajo (2011), who address the task of translating instructions into formal PDDL plan descriptions. They use an HTML parser to extract relevant sentences from a instructional web page, which are then parsed by an NLP parser. Although the details of this last parser are not given, it is known that it makes use of standard NLP approaches and tools such as stemming, stopwords and WordNet (Miller 1995). The result of this NLP parser is the labelling of a number of subsets of each string, such as the unit of measure of an ingredient, or the identification of the main verb that describes the action that should be performed in a step. These

[8]`https://twitter.com/` (accessed on 17/10/2017)

labels are then represented as a PDDL actions. For example, the step “Mix flour and salt” could be translated into the action signature `mix(flour,salt)`.

The results of this early work provide evidence of the difficulty in parsing natural language expressions about instructions. The authors focused their evaluation on three categories of instructions, including cooking recipes, and reported 68% accuracy of the extracted PDDL actions. Moreover, representing the details of natural language expressions as PDDL actions appears to be restrictive, as complex expression are reduced to a set of actions and components. It is unclear how other aspects of a step, such as time constraints, modalities, optional actions and vague terms can be parsed and represented in PDDL.

A higher accuracy was reported by Jung et al. (2010), who reached 84% accuracy in the task of generating a “situation ontology” based on the concepts of “goals”, “actions”, “objects” and “locations” from wikiHow and eHow instructional articles. To extract these concepts an NLP approach is tried first, namely syntactic pattern-based parsing, and then, in case of failure, a ML classifier using the conditional random field method is used. The higher precision achieved by this system compared to the work by Addis and Borrajo (2011) can be seen as an indication that the goal of extracting ontological concepts from a instructional natural language labels is easier than the task of converting them into AP expressions.

An accuracy of 80% of the extracted knowledge was reported by Kiddon et al. (2015) in parsing cooking recipes. Their approach relies relies on NLP parsing to label words into a number of categories, such as “food” or “location” which are then fed to an expectation-maximization ML algorithm to construct an *action graph*. This graphs defines which actions should be performed on which items and in which order. The focus of this work on the cooking domain is not unusual, as this domain can be regarded as one of the simpler to analyse and most researched domains of human know-how. Other applications of NLP in the cooking domain have been developed by Schumacher et al. (2012), Malmaud et al. (2014) and Tenorth et al. (2010). Other domains of human know-how related to areas such as social interactions, animals, philosophy, religion and arts are often not considered. The main difference with the cooking domain is that these other areas cannot be easily abstracted into a list of actions and objects.

Tenorth et. al. for example, parsed wikiHow cooking instructions into executable robot plans. Their approach combines the NLP parsing of natural language instructional texts with an existing knowledge base of actions and camera-identifiable objects. The ambitious goal of automating an entire recipe using a robot was achieved by supplying the robot with additional knowledge and domain specific information. In particular, human recipes tend to contain very high level descriptions of actions where many details are left implicit. In order to achieve robotic automation, however, the robot tried to infer these details and, when not possible, they had to be supplied by humans. Overall, this work provides insights on the amount of common sense information that is omitted from human instructions, although being necessary for their execution.

Another system based on NLP to extract procedural knowledge from instructions has been performed by Zhang et al. (2012). This work used NLP parsing and extraction grammars to detect the various components of an instructional sentences, such as the action, the object it is applied on, the instrument it is used and the purpose of the action. This system detected the different components of a set of instructions with different levels of accuracy, ranging from 68% to 97.3%. These results seem to suggest that certain components of a set of instructions are easier to detect than others. For example, the main actions and temporal parameters seem to be easy to detect, while instruments and spatial parameters are not. Zhang et. al. acknowledge the importance of extracting procedural knowledge that can support multiple applications. They speculate a number of such applications although the suitability of their formalisation in the context of those applications was not tested.

These studies demonstrate the wide variety of possible approaches to extract procedural knowledge from instructional texts. However, their results cannot be directly compared with each other as they were tested on qualitatively different datasets and adopted different evaluation approaches. More importantly, a direct comparison is not possible because each approach extracted different aspects of instructional know-how which, in many cases, were not formally defined. Without a shared semantic representation of instructional know-how, these approaches cannot be directly compared and integrated.

Overall, these approaches do not demonstrate the reusability of the extracted knowledge across multiple applications. Moreover, these approaches inherently suffer from a significant decrease in accuracy compared to the original sources. Some of these approaches are also limited by their domain-specificity or by formalising an incomplete representation of know-how where certain aspects are missing. It can therefore be said that a domain-independent approach to extract reusable instructional knowledge without a significant loss of accuracy is not currently available in the literature.

2.3.2 Know-How Acquisition without Instructions

Instructions are not the only class of web resources that contain procedural knowledge. A number of projects have investigated the possibility to extract know-how from non-instructional resources. For example, Fukazawa and Ota (2010) developed a hierarchy of tasks to help mobile users navigate a space of possible services that can be used to achieve the users' goals. This hierarchy is generated by analysing frequent verb-object associations, such as "watch" and "movie" from the context words of the results of a web query for a given topic, such as "movie". The chosen topics have been taken from 31 content categories of Yahoo! Japan.[9] This work shows how statistical features such as the correlations of verbs and nouns in a corpus of documents can be used to infer tasks and their sub-activities even in the absence of an explicit representation of instructions.

In the medical domain, Song et al. (2011) analysed the abstracts of biomedical scientific publications to populate an ontology with medical-specific procedural know-ledge. In their work, a procedure is composed by a *method*, such a treatment or a medication, that is applied using a particular *action* onto a *target*, such as a disease or an organ. A complex procedure, called *purpose*, can be decomposed into a set of simpler ones, called a *solution*. The abstracts of the chosen scientific publications was preprocessed using NLP tools such as POS tagging and syntactic parsing. The output of this preprocessing step were then fed to ML classifiers to identify the components of the various processes in the text. Overall, their evaluation shows an accuracy of 81% in identifying the purpose and solution described in an abstract, and of 62% in identifying the target, action and method of a particular process.

When following a set of instructions on how to do a task, the completion of such task is typically the main outcome that is being pursued. However, tasks might also have important secondary outcomes. Preparing and eating certain foods, for example, could impact the sport performance of an athlete on the following days. Discovering this type of outcome knowledge is the focus of the work by Kıcıman and Richardson (2015). In their work, the association between tasks and outcomes is mined on Twitter and it is based on the assumption that users are likely to tweet about the discovered outcomes of certain activities soon after tweeting about performing such activities. To achieve this goal, a number of problems was addressed, such as identifying tweets about a user's experience, estimating the time of the user's experience (which might be

[9] `http://www.yahoo.co.jp/` (accessed on 17/10/2017)

different from time when the tweet was published), linking user's experiences together into timelines of related events and classifying outcomes of events as either positive or negative. The evaluation of this system was performed on two case studies, namely running a marathon and adopting a pet. While it is unclear how well this system would generalise to other scenarios, the results obtained in these case studies suggests that information about common outcomes of certain actions can be extracted from Twitter data.

2.3.3 Learning Programs from Non-Programmers

While instructional resources are a convenient source of readily available know-how, this type of knowledge can also be acquired directly from humans. Research in this direction has been conducted by Azaria et al. (2016), who have developed an instruction learning system capable of learning automatable commands from users within the email domain. These commands are communicated in natural language by a user through a dialogue interface. This interface is used both to learn procedures, such as sending an email, and to learn relevant factual data, such as the email addresses of certain individuals. Commands defined by a user can in principle be shared and reused by other users, although the possible extent of this reuse is not clear. The experimental evaluation of their system reports that 41% of the test subjects managed to complete the whole experiment and that 85% of the defined commands could be parsed correctly.

A similar approach was developed by Quirk et al. (2015) to parse "if-this-then-that" natural language statements into executable code. An example of "if-this-then-that" instruction in natural language is "send me an email if it is sunny in Edinburgh". This instruction would be implementable by a service capable of monitoring the current weather forecast and one capable of sending emails. The evaluation of their system reports a 81% accuracy in parsing "if-this-then-that" statements that are unambiguous to humans.

Branavan et al. (2010) created a system that can automatically learn how to execute instructions in the domain of GUI interactions with an operating system. In this restricted domain, the actual steps required to complete a task such as "open control panel" can be learnt by assuming (1) a limited set of possible actions, such as user clicks on the interface, and (2) a way to determine whether a task has been accomplished successfully. This system learns how to execute instructions using reinforcement learning and without relying on an explicit model of the environment. Certain aspects of the model are, on the other hand, learnt dynamically. Domain knowledge is injected in the description of the states of the environment and in the choice of the available primitive actions (or low-level actions), such as the action "double click". Domain knowledge is also required to define the reward function used by the reinforcement learning algorithm, which assumes a correlation between the words found in the instructions, such as "control panel", and the labels of certain objects which should be available in desirable states of the environment associated with high reward. The method described by Branavan et al. (2010) relies on several domain-specific assumptions and it is not clear to what extent it could be applied to other domains.

Further proof of the feasibility of acquiring formal instructional knowledge directly from users comes from websites that offer *automation services*. Automation services allow users to create (and often share) automated workflows that combine several applications together, such as email providers and web-storage. For example, If This Than That (IFTTT)[10] allows users to create simple fully automated workflows where a trigger, such as receiving an email, triggers a certain action, such as the creation of a file in an online repository.

A similar product is Microsoft Flow,[11] although it is more oriented toward the integration of Microsoft products. Tasker[12] is an application similar to IFTTT that allows users to create simple workflows combining Android's applications and triggers. While allowing for more complex workflows than simple triggers, it is also aimed at a more technical audience that is willing to spend more time learning how the system works to achieve better automation. Similar to Tasker is Zapier.[13] Zapier can be used to define more complex workflows than IFTTT by allowing multiple effects to be joined to the same trigger.

Although typically offering some of their functionalities for free, these automation services are commercial systems. While they focus on integrating different applications and services, they do not release data about the workflows in a way that is reusable by and integratable to other systems. Two other important limitations of such systems are the very limited set of workflows operators allowed (usually only one trigger and one action) and the inability to represent manual tasks, which limits their applicability to fully-automatable workflows only.

[10]`http://ifttt.com/` (accessed on 17/10/2017)
[11]`https://flow.microsoft.com/` (accessed on 17/10/2017)
[12]`http://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm` (accessed on 17/10/2017)
[13]`http://zapier.com/app/explore` (accessed on 17/10/2017)

2.4 Applications of Instructional Knowledge

A machine-understandable representation of human instructions can be applied in a wide variety of areas ranging from Information Retrieval to Service Recommendation (Myaeng et al. 2013). Two examples of areas that can benefit from formalised instructions are Activity Recognition and Robotics.

The central problem addressed by Activity Recognition is the identification of a top level activity (or intention) from a set of observations (Kim et al. 2010). The intention of preparing tea, for example, could be inferred after observing an agent boiling water and placing a tea bag in a cup. A challenge faced by Activity Recognition systems is the acquisition of a model of the activities to be recognized. If the domain of interest involves common human activities, their model can be extracted from web instructions and used to recognize such activities (Perkowitz et al. 2004).

While the application of human instructions to Robotics remains an ambitious goal, one experiment in this direction was conducted by Tenorth et al. (2011) and employed a robotic agent. The agent attempted to perform the activity of preparing a pancake by following user-generated instructions retrieved from a wikiHow website. This experiment explored the potential of human know-how for process automation and highlighted the importance of integrating individual processes with external sources of knowledge. Artificial agents, in fact, require more information about a process than what can typically be extracted from a single set of instructions, as they lack human common sense. Knowledge about an ingredient, for example, allowed the agent to learn what it looked like and whether, by virtue of being perishable, it might be found in the refrigerator.

These applications demonstrate some of the possible uses of formalised instructional knowledge. However, such applications rely on domain specific formalisations, which are optimised for the task at hand, but that cannot be easily re-used. Overall, the possibility to use a single formalisation of instructions to support a diverse range of applications has not yet been demonstrated.

Chapter 3

Methodology

The main problem addressed in this thesis is the inability of machines to understand instructional resources on the web. As discussed in Chapter 1, I address this problem by simultaneously addressing three main sub-problems, namely know-how representation, know-how acquisition and know-how applications. In other words, after developing a knowledge representation formalism suitable to modelling the domain of human instructions (know-how representation), I proceed to inquire whether such representation could be extracted automatically (know-how acquisition) and whether it is sufficiently rich to support human exploration and execution of instructions (know-how applications).

To the best of the author's knowledge, no previous research has addressed all three of these sub-problems within a unified framework and there is no single methodology in the literature that can be used to address them. Although correlated, these three problems require qualitatively different solutions and distinct methodologies for their evaluation. For example, knowledge acquisition approaches can be evaluated by testing them against real-world benchmarks, while the potential applications of a knowledge base tend to be demonstrated by the development of prototypes. In this chapter I present the different methodologies that I use to evaluate my solutions to the three problems mentioned above.

3.1 Know-How Representation Methodology

The central contribution of this thesis, which binds together all the other contributions, is the use of a common knowledge representation of the domain of human instructions. Before one such representation can be adopted, or created if it doesn't exist, it is fundamental to formalise its *knowledge requirements*, or in other words, what is the type of knowledge that should be captured. This formalisation features prominently, for example, in the ontology development methods reviewed by Jones et al. (1998). Knowledge requirements are not only useful to test the suitability of existing knowledge representations, but can also be used to guide the development of one such representation. As such, they feature both in literature about ontology development (Suárez-Figueroa et al. 2012) and on ontology evaluation (Vrandečić 2009).

A common approach to defining knowledge requirements, found for example in the recommendations by Noy and Mcguinness (2001), is through the formulation of *competency questions* (Grüninger and Fox 1995). Competency questions define, either formally or informally, the type of queries that a knowledge base should be able to answer. Since the use of competency questions for the definition of knowledge requirements has been widely adopted in recent years, for example by Suárez-Figueroa et al. (2012) and Ren et al. (2014), I have chosen this approach for the creation of a knowledge representation of the domain of human instructions. I will describe this methodology in Section 3.1.1.

Ensuring the satisfiability of the knowledge requirements is a crucial phase in the definition of a knowledge representation. However, different representations can be used to satisfy the same knowledge requirements, and other criteria can be considered. A common suggestion, found for example in the methodologies developed by Fernández-López et al. (1997) and Suárez-Figueroa et al. (2012), is to reuse as much as possible the terminology already used in the domain. This process allows both formal and informal terms used in a domain to be directly mapped to the terms used in the ontology. This makes the ontology more understandable by experts in the domain, and generally more interoperable with other resources in the same domain (Suárez-Figueroa et al. 2012). The knowledge representation discussed in this thesis aims to close the gap between human and machine understandable representations. It is therefore desirable to reuse, and potentially combine, the terminology used by humans when communicating instructions, and the terminology used by machines to represent formal processes. I will describe this approach in Section 3.1.2.

These methodologies provide a principled approach for defining a formal representation for the domain of human instructions. After this representation is available, it is possible to test whether it can provide a solution to the central problem addressed in this thesis by testing it in the remaining two sub-problems, namely know-how acquisition and know-how applications. I discuss whether this type of representation can be extracted automatically from existing resources and whether it can support human exploration and execution of instructions in Sections 3.2 and 3.3 respectively. As pointed out in Section 1.1, these two problems have conflicting requirements on the semantic richness of the representation. By addressing both problems with the same knowledge representation, I aim to demonstrate that the chosen representation is simple enough to be acquired automatically yet expressive enough to enable machine automation in novel applications.

It is important to mention that in order to acquire knowledge of a domain, ontology development methodologies such as the one proposed by Fernández-López et al. (1997) usually involve the intervention of domain experts. This intervention is necessary when the domain experts and the ontology experts are separate individuals. However, given that the domain of human instructions is not a specialised domain, there is no specific set of people that could be considered as authoritative domain experts. Concepts found in human written instructions, such as “steps” and “requirements”, are not formally defined and they are usually based on common sense knowledge. It should be noted that the meaning of these concepts adopted in this thesis is based on my interpretation and on that of those who collaborated with me on this project. Our interpretation has been informally validated by exposing it to a wider audience, such as the research community, and by running experiments which included crowdsourced human judgements without exposing discrepancies between our interpretation and the interpretation of others. Given the simplicity of the domain, and this preliminary evidence, I decided that further experiments to validate the generality of our understanding of this domain, although useful, were beyond the scope of this work.

3.1.1 Evaluating the Requirements of the Knowledge Representation

The satisfiability of a competency question with respect to a knowledge representation can be proven mathematically (Department and Gruninger 1996). In practice, however, the potentially large number and complexity of the competency questions can make such an approach infeasible. This is particularly problematic for *iterative-incremental* ontology development life-cycles (Suárez-Figueroa et al. 2012), where competency questions might be modified and further refined across multiple iterations. In such cases, an experimental evaluation of the competency questions seems preferable. For example, following similar principles to unit testing (*Method 19* in Vrandečić 2009).

On this approach, a computational method, such as a database query, is created to answer each competency question. The results computed by this method on a number of test cases are then compared with the predefined set of expected answers; see for example the OntoEdit tool developed by Sure et al. (2002). In this thesis, I will evaluate the satisfiability of the competency questions using this experimental approach and discuss the tests I performed. Good testing practices will be followed, in particular by testing each competency question in a set of comprehensive test cases.

3.1.2 Evaluating the Reuse of Domain Terminology

When defining a formal representation of a domain it is a good practice to reuse as much as possible of its existing terminology. I do this through a review of both informal and formal representations of instructional knowledge. For each such representation, I will review the terms which are relevant to the domain of human instructions. It is important to notice that this review aims to identify the *common* terms that can be used to represent instructions, and it is not meant to be exhaustive.

To review informal representations, I search for the main instructional websites which contain domain-independent semi-structured textual instructions in English. I conduct this search in two ways. The first is a keyword-based search of such websites using a web search engine. The second is an analysis of existing reviews of instructional websites. To decide which are the most important instructional websites, I consider factors such as the number of instructions they contain, the number of users, and their prominence within search engine results when relevant search queries are asked.

In the second part of my review, I search for existing approaches that can be used to formally represent instructions. Such formal representations feature in a large number of different research and application fields. The first step of this review is therefore the identification of the main fields that feature knowledge representations that are applicable to human instructions. In the second step, I review the terminology found in specific approaches within the chosen fields. Given the large number of different fields that feature formalised instructions, an exhaustive review of all the approaches would be impractical. Instead, I focus on the most important approaches in the field, chosen by virtue of their prominence in their field, and considering factors such as the number of papers that describe them and their overall impact in terms of citations. Concepts which are found in prominent papers found within relevant research areas will be assumed to be common terms that should be considered for reuse. However, due to the lack of a thorough review of all the approaches in all of the relevant fields, common terms used to describe processes other than the ones discovered by this method might well exist.

3.2 Know-How Extraction Methodology

A common bottleneck for knowledge representations is the difficulty of populating them with information about real-world entities. This process, sometimes called *ontology population* (Maynard et al. 2008 and Faria et al. 2014), *knowledge acquisition* (Welty and Murdock 2006) or *knowledge extraction* (Gangemi 2013), typically relies on manual efforts and sometimes on the availability of expert intervention. One of the main drawbacks of such manual approaches is the difficulty in extracting knowledge from large amounts of avaliable resources. This problem has led to the development of automatic general-purpose knowledge extraction tools, such as the ones reviewed by Gangemi (2013). However in domain-specific scenarios, such as medical procedures (Song et al. 2011), ad hoc solutions might be required.

The know-how acquisition problem addressed in this thesis requires an effective method to extracting knowledge from existing sets of instructions. I show this through an experimental evaluation of a knowledge-extraction system. More specifically, this knowledge extraction involves two steps: (1) the extraction of a formal representation of instructions from existing web resources and (2) the integration of these representations into a unified knowledge base. I describe my methodology for evaluating these two steps in Sections 3.2.1 and 3.2.2 respectively. The usefulness of a knowledge base also depends on how effectively it can be shared and reused. I describe a methodology to evaluate this last aspect in Section 3.2.3.

3.2.1 Evaluation of the Know-How Extraction Systems

The results of a knowledge extracting system can be evaluated in terms of how *accurately* they represent the original knowledge contained in the sources they are extracted from. Accuracy is often defined as a combination of *precision* and *recall*. In informal terms, precision is a measure of what proportion of extracted facts are correct. A higher level of precision allows us to be more confident of the truthfulness of the extracted facts. Recall is a measure of how much of the original knowledge has been extracted. A higher level of recall allows us to be more confident that facts contained in the original sources have been extracted.

In the human instructions domain, the evaluation of a knowledge extraction system involves a comparison between the original web resources that describe instructions, such as HTML pages, and the model of these instructions that results from the extraction. Human judgement might be required to compare the knowledge contained in these resources. The method I propose for this kind of evaluation involves collecting human judgements. More specifically, humans can be asked to provide answers for a number of questions about a set of instructions. Example of possible questions are: "what are the steps of the set of instructions x?" or "what are the requirements of task y?". I propose to use the competency questions described in Section 3.1 for this evaluation, since they are meant to capture the main types of knowledge that a knowledge representation should formalise.

Two groups of informants should be considered, namely those that answer these questions given the original sources, and those that provide answers given a human understandable visualisation of the extracted knowledge. Precision and recall measures can then be calculated by comparing the answers given by these two groups. The more the answers of both sets are similar, the more evidence we have that the accuracy of the extraction systems is high, with respect to the considered questions. The main drawback of this evaluation is that it does not consider other types of knowledge that are not captured by the chosen set of questions. For this reason, other types of knowledge might exist that are not captured by the extraction system.

3.2.2 Evaluating Know-How Interlinking Systems

After extracting the representation of a number of sets of instructions, these representations need to be integrated into a coherent knowledge base. This integration is needed to determine which elements are the same, or how they relate to each other. To perform this integration I will develop an interlinking system capable of automatically discovering relations between the elements of the sets of instructions.

To evaluate the quality of the results of the know-how interlinking system, I will compare the links automatically created by this system (population A) with the links already existing in the set of resources I considered which are considered to represent the same semantic relation (population B). The links in these populations can be seen as ordered pairs of elements. For example, a human might have created an HTML link between two instructional web pages to indicate that more information about one element of one set of instructions is provided by the other set of instructions. The semantic interpretation of these links will be based on the official guidelines on how to create such links published by the website hosting the links. Such guidelines will be assumed to be followed by the user base of the website. A study on the extent to which users follow these guidelines when creating links, however, has not been performed.

I will compare these populations using the same two metrics as with know-how extraction, namely precision and recall. Precision will measure whether the links are *correct* or not according to their semantic interpretation. For example, we can imagine an interpretation of a link as a connection between a set of instructions and the output that is generated by performing the set of instructions. Under this interpretation, a link between a set of instructions on "how to make a pancake" and the object "pancake" should be considered correct. A link between the same set of instructions and the object "car", instead, should be considered wrong. This type of judgements will be collected using human evaluation.

This manual evaluation will be performed only on a randomly selected subset of links for each population because the size of the populations of links is expected to be very large. These manual evaluations of a sequence of randomly selected samples from a population, each one resulting in a boolean judgement (correct or not correct), can be modelled as a binomial distribution. The probability that one selected element is classified as correct is interpreted as the precision metric that is to be evaluated. For this reason, I will use the chi-squared test (Powers 2011) to determine whether there is a significant difference between the two populations with respect to the precision measure.

The recall metric is intended to measure the ratio of all the possible correct links that is found within a population. Unfortunately, determining all the possible correct links is not possible, due to the very large number of possibilities to consider, and the absence of a gold standard. For this reason, my analysis will be limited to determining which population has the highest recall by estimating the number of correct links it contains. Since the number of all the possible correct links will be the same for both populations, as they refer to the same set of instructions, the population with the higher number of correct links will be considered to have higher recall. The number of correct links found within a population can be computed as the product of the precision measure of the population with the number of links it contains.

3.2.3 Methodology to Evaluate Linked Open Data Compliance

Linked Open Data represents the most prominent approach to publishing reusable semantic data on the web. I will demonstrate that the methods I developed constitute a sufficient pipeline to publish Linked Open Data in the domain of human instructions. To evaluate this claim, I will evaluate my approach against the 5-Star Open Data ranking system,[1] which defines 5 necessary and sufficient criteria to determine whether a dataset complies with the Linked Open Data principles. Four of these criteria can be satisfied by well-defined qualitative properties, such as the adoption of a particular data format. Compliance with these four criteria can therefore be easily validated by demonstrating that my set of methods possesses these qualitative properties.

The last criterion of this ranking system relies on a quantitative measure by requiring the dataset to be *linked* with other datasets. While the concept of link (or *RDF link*, Bizer et al. 2009a) is well defined within Linked Data, the 5-Star Open Data ranking system does not specify a measure of *how linked* a dataset should be. This last criterion is mostly intended to convey the principle that concrete efforts should be spent in linking datasets, and an exact threshold does not exist because it would be difficult to define for all possible scenarios. To evaluate the compliance of my methods with this last criterion, I will consider whether a *significant* number of links can be generated to an *important* public dataset. I will consider DBpedia as an important public dataset, given its importance within the Semantic Web community and its centrality within the Linked Open Data Cloud,[2] a graphical representation of the major interconnected Linked Data datasets. To determine whether a *significant* number of links have been found, I will evaluate whether at least one link is found for every set of instructions. I have chosen this threshold based on an arbitrarily judgement, given the lack of more precise guidelines applicable to this domain.

3.3 Know-How Applications Methodology

The main methodology I adopt to demonstrate the possible uses of formalised instructional knowledge is the development of prototypes. The development of prototypes aims to demonstrate the feasibility of developing applications with a particular set of features. I will discuss such features, and in case similar systems have already been developed, like in the case of know-how exploration, I will clarify what new features

[1]`http://5stardata.info/en/` (accessed on 17/10/2017)

[2]`http://lod-cloud.net/versions/2017-02-20/lod.svg` (accessed on 17/10/2017)

the prototype offers. Contributions 5, 6 and 7 are evaluated using prototypes.

It should be noted that prototypes tend to be, by their very nature, incomplete and unpolished systems; more suitable for demonstration purposes than for real deployment. For this reason, I will not evaluate prototypes on the basis of quantitative features such as usability or user rating. Instead, I will evaluate them on qualitative features. More specifically, I will focus on functionalities that are novel and that cannot be replicated by existing systems.

A concrete implementation that successfully replicates the desired features provides evidence that, within the scope of the given test conditions, the desired features of the application are achieved. One limitation of this methodology is that the results obtained by testing the prototype are not guaranteed to be replicable on different real-world deployments of similar systems. The extent to which the results obtained by testing a prototype are generalisable can be increased by performing large-scale evaluations in a diverse range of significant conditions. However, due to the practical constraints derived from the need to test multiple such applications, smaller scale tests and prototypes will be preferred.

3.3.1 Evaluating New Features for Know-How Exploration

A common application of formalised knowledge is supporting knowledge exploration. An example of this is Google Knowledge Graph,[3] which is a collection of structured knowledge that is used to display more informative results to the users of a web search engine. Formalised knowledge enables a user to achieve certain exploration goals in qualitatively different ways. For example, a user might have the goal of discovering the birth date of an important historical figure. Using a naive search engine, the user will have to first search for documents related to the historical figure, and then have to manually search this document to find the desired piece of information. Using formalised knowledge, however, the web search engine can immediately provide to the user a number of facts about this historical figure, which might include the birth date. In this example, the two approaches to achieve the user's exploration goal are qualitatively different. In fact, the second approach does not require the user to retrieve and inspect a web resource.

To evaluate the applications of a formal representation of instructions I will formulate a number of exploration goals, and discuss their importance in the context of

[3]`https://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html` (accessed on 17/10/2017)

discovering know-how. Then, I will demonstrate through the use of a prototype that they can be achieved more efficiently by using the prototype than by using the original websites where they were found. To do so, I will analyse the type of users' actions required to achieve an exploration goal using a certain system. These actions will belong to a category, such as clicking links, scrolling through options, or reading paragraph of text. A system A will then be said to be more *efficient* than a system B in achieving a certain goal X if the actions required to achieve the goal in system A are a subset of those required by system B. This notion of efficiency assumes that users have an interest in achieving an exploration goal with as few actions as possible.

It should be noted that this type of analysis does not consider quantitative differences between the same type of actions. More specifically, I will make the simplifying assumption that all actions in the same category are equal. In reality, however, this might not be the case. For example, two clicking actions might have different levels of usability, as users might prefer clicking a larger and more visible link than one which is small and harder to see. However, I consider such differences as outside the scope of this project by making the assumption that they do not depend on the underlying knowledge representation used, but on the details of the user interface instead.

3.3.2 Evaluating a Method for Decentralised Know-How Exploration

Knowledge on the web is spread across multiple resources, and human expertise is typically needed to find and integrate related knowledge. For example, the solution to a complex problem might require the integration of multiple sets of instructions. Formalised knowledge about instructions can be used to replicate part of this human capability. However, typical approaches to navigate formalised knowledge assume the availability of a centralised knowledge base.

In Section 7.2 I propose an approach to allow machines to search and integrate web data dynamically. To evaluate whether such approach allows decentralised exploration of instructional knowledge, I define the fundamental requirements of the approach. From the definition of these requirements it should be possible to infer that a system that fulfils these requirements can achieve decentralised exploration of instructions.

A full formalisation of this system to mathematically prove this hypothesis is outside the scope of this thesis. Instead, I will evaluate the hypothesis using a semi-formal description of my approach and demonstrate how it supports the goal of decentralised exploration. I compensate for this lack of formality, which is a potential danger for ambiguity, with a practical implementation of this approach. To do so, I will create a prototype and experimentally verify that it enables the intended goal.

3.3.3 Evaluating a Method for Know-How Execution

An ambitious use of formalised human instructions is to allow machines to actively collaborate with humans in the execution of such instructions. In particular, I will focus on sets of instructions that cannot be fully automated but might benefit from partial automation. For example, a human could write a set of instructions on "how to organise a workshop". While certain steps of this process should arguably be done by a human, such as the choice of the topic of the workshop, other steps could be automated, such as the dissemination of information about the workshop through different channels.

Assuming that a formal representations of human instructions is available, can a know-how execution system use it to support collaboration between humans and machines? The evaluation of such a system is not trivial, and addressing it involves demonstrating the existence of a technical solution to develop such a system, and demonstrating that it can be used in practice.

I demonstrate the existence of a technical solution by proposing a concrete method that, given a formalised set of instructions, enables human and machine to collaborate in executing that set of instructions. This method can be tested by implementing it in a prototype. Once a prototype has been developed, its applicability in practice can be evaluated.

The practical applicability of such a system depends on several of factors. An important factor is the number and complexity of tasks that the system can automate. A system that can automate a large number of complex tasks is likely to be more useful than one that can only automate a small number of trivial operations. A related factor is the domain of the set of instructions. Instructions in certain domains, such as computers and technology, might contain significantly more automatable tasks than other domains, such as social know-how. For example, we might expect to find more automatable steps in the set of instructions on "how to update your operating system" than in the steps of the set of instructions on "how to find inner peace". Another relevant factor is the user interface for interacting with the system. Different interfaces might make it easier or harder for humans to write instructions and collaborate with machines.

A thorough analysis of all these factors would be a major undertaking and it is beyond the scope of this thesis. For this reason, I limit the evaluation of the practical applicability of the prototype to a single scenario. That is, I define a single task and create a number of automatable functionalities that should be relevant to this task. Then, I evaluate whether humans can successfully create a set of instructions to achieve that task that can be executed both by humans and machines using the prototype.

To perform this evaluation, I describe the task to a number of human participants, and then ask them to create a set of instructions on how to achieve the task. The practical applicability of the prototype in a practical scenario will be considered successful if the contributions provided by the participants result in instructions that can be partially automated by the prototype. Human participation in this experiment is gathered using a crowdsourcing platform.

This type of evaluation will provide preliminary insights into the challenges involved in developing this type of instruction-based collaboration between humans and machines. If successful, it will demonstrate that this type of collaboration is possible in a specific scenario. Obviously it does not provide information on the generality of this approach and on the extent to which it would be applicable to other types of instructions, but nevertheless seems adequate as a proof of concept.

3.4 Conclusion

In this chapter I have described my methodology to address the three main problems discussed in this thesis, namely know-how representation, know-how acquisition and know-how applications. The significant differences between these problems result in the need for significantly different methodologies. In the following chapters I will adopt these methodologies to evaluate the concrete hypotheses that constitute the main findings of my work.

Chapter 4

Know-How Representation

In order to represent procedural knowledge, a suitable knowledge representation formalism needs to be adopted. Before moving to the details of the formalisation, I will describe in abstract terms what this formalism should conceptualise, or in other words, an abstract model of human-made procedures and their execution. This formalisation should be able to describe human *step-by-step* instructions and their potential execution following a *to-do checklist* approach.

Two main aspects appear to be common in step-by-step instructions. The first one is the decomposition of tasks into simpler ones, and the second is the organisation of tasks into workflow structures, such as a sequential ordering of steps. These aspects can be consistently found across all of the related work on instructional knowledge acquisition reviewed in Section 2.3.1.

Throughout this thesis, I will use the term *task* to refer to "something that can be accomplished". In this context, *step-by-step* instructions, such as instructions on "how to prepare a pancake" can be seen as explanations of how to achieve a particular task, such as "preparing a pancake". This task can be said to be the *goal* of the set of instructions. The goal of a set of instructions is typically decomposed into a set of smaller sub-tasks, or steps, such that the completion of some or all of these sub-tasks is equivalent to the completion of the main task. The term *step*, therefore, refers to tasks that are used to decompose other tasks. Different ways to decompose the same task might exist, thus representing different alternative ways, or *methods*, to accomplish a goal. Moreover, sub-tasks might need to be executed following a certain total or partial ordering. This ordering can be thought of as dependencies between the tasks.

Regarding the execution of a set of instructions, there is no single approach universally adopted by humans. In order to limit the complexity of the problem, the notion of executing instructions in this thesis is limited to a single approach, namely following a *to-do checklist*. The execution of a to-do checklist to complete a task, or goal, x can be seen as the process of assigning the value of *checked* to previously *unchecked* sub-tasks of x once they have been executed and completed. This process is generally continued until their combined completion can be considered equivalent to the completion of the goal task x, which is then also marked as checked. The process of completing a checklist is visualised through a set of check-boxes, which represent the execution of a specific sub-task. As a check-box can be checked or unchecked, the execution of task can have the property of being accomplished or not.

Tasks can be accomplished multiple times, and the concept of a specific checklist can be seen as the context, or environment, in which tasks are executed. For example, the act of repairing the tire of a car is only relevant to a specific car, and only within a limited time period. After some time, the tire might get punctured again, and the same process of fixing it might have to be repeated. Therefore, multiple checklists to complete the same task might exist.

This simplified notion of execution has been chosen for three main reasons. The first one is the commonality of this approach. It can be argued that achieving a goal by following a checklist is a commonly used and well understood concept. The second reason is the simplicity of its representation. The level of completion of a task is simplified to a boolean value: either complete (checked) or incomplete (unchecked). The last reason is that this approach involves explicit logging of the execution state.

Information on the current state of the execution of a set of instructions is a crucial piece of information that is needed, for example, to determine which task should be performed next. Humans often complete instructions without the need to explicitly log the state of the execution. When following simple instructions, it might be possible to determine the current state of execution just by resorting to memory, or by looking for cues in the environment. For example, a pot of boiling water can remind us that the action of boiling the water has already been performed.

Machines, on the other hand, do not have direct access to the user's memory. While machines could use sensors in an environment to determine the state of the execution, for example by trying to recognise activities using computer vision (Ke et al. 2013), this process would require dedicated sensors and sophisticated AI processing. This problem can be avoided if users explicitly log the state of the execution using a checklist approach. Checking items on a checklist results in a small overhead for the users. Nevertheless, it is a popular technique as it allows humans to recall the state of the

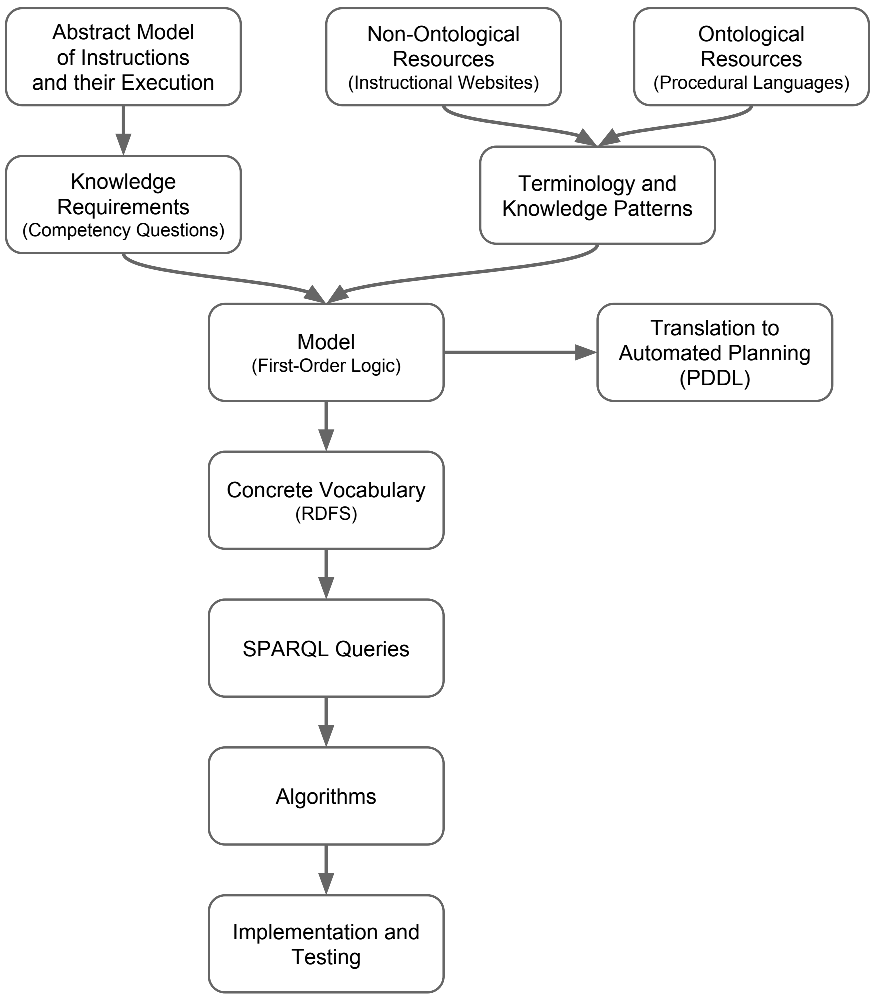

Abstract Model of Instructions and their Execution
Non-Ontological Resources (Instructional Websites)
Ontological Resources (Procedural Languages)
Knowledge Requirements (Competency Questions)
Terminology and Knowledge Patterns
Model (First-Order Logic)
Translation to Automated Planning (PDDL)
Concrete Vocabulary (RDFS)
SPARQL Queries
Algorithms
Implementation and Testing

Figure 4.1: Overview of the various topics discussed in this chapter and how they feed into each other.

execution at a later time, or to easily communicate it to external collaborators.

Instead of describing human instructions and their execution separately, I choose to combine them in a unified model. This unification allows me to provide definitions of the various components of a set of instructions with respect to their execution. For example, it is possible to define what it means for something to be the “step” of a set of instructions by defining how this information should affect our decisions on how to execute that set of instructions.

4.1 Chapter Overview

Figure 4.1 illustrates the various components of this chapter. In the introduction, an abstract model of the domain of instructions and their execution was described. This abstract model is then turned into a set of knowledge requirements (Section 4.4). An analysis of ontological and non-ontological resources (Sections 4.6 and 4.5) related to this domain is then integrated into a set of common terms and knowledge patterns. This information is then used to create a formal model that satisfies the predefined knowledge requirements (Section 4.7), which in turn is materialised into a concrete vocabulary (Section 4.8). The proposed model is also translated into an existing planning language (Section 4.9) to show how it can be mapped to pre-existing formalisations. A set of queries defined over the vocabulary is used in combination with a number of algorithms to validate the predefined knowledge requirements. Full details of these queries and algorithms is given, respectively, in Appendices C and D. Finally, these algorithms and queries are implemented and tested to verify their correctness.

4.2 Problem Description

The general hypothesis proposed in this chapter is that a knowledge representation formalism, sufficiently expressive to model the instruction and checklist domain, can be described with concepts which are common both in the informal representations of this domain created by web users and in those created by experts to represent processes. This general hypothesis can be split into two more specific hypotheses.

The first hypothesis addresses the question of whether the proposed model is sufficiently expressive to model this domain. It is formulated as follows:

Hypothesis 1. *The competency questions that define the knowledge requirements of the instructions and checklist domain are answerable by the proposed knowledge representation formalism.*

The proposed knowledge representation formalism will be described in Sections 4.7 and 4.8. The competency questions that define the knowledge requirements for this domain will be explained in more detail in Section 4.4. This hypothesis will be validated in Section 4.10. Although there is no definitive measure of how well a set of knowledge requirements enables a formalism to be used in a domain, an argument as to why the selected requirements are relevant to the instruction and checklist domain is presented in Section 4.4.

The second hypothesis is formulated as follows:

Hypothesis 2. *The proposed knowledge representation formalism uses concepts that are common in instructional websites and in formal process description languages.*

Evidence in favour of the second hypothesis will be presented in Sections 4.5 and 4.6. The knowledge extraction performed on instructional websites and described in Section 5.1.1 can be considered as further evidence of the similarity of the proposed model with web instructions. Moreover, the translation of the proposed formalism into a standard planning language described in Section 4.9 can be considered as additional evidence of its compatibility with pre-existing process description languages.

4.3 Domain Formalisation Method

When it comes to formalising a domain, many choices can be made. Guiding principles on how to make these choices come from the field of Ontology Engineering, such as the methodology developed by Sure et al. (2009). The formalisation of a domain, i.e. an ontology, can be seen as a component of the Ontology Engineering life-cycle. This component typically involves activities such as formulating high-level specifications, acquiring domain knowledge, and developing progressively more formal models until an ontology is made. This can be a very cyclic process, where mistakes or newly found pieces of information can cause the process to return to previous phases.

In this thesis, I have chosen to formalise this domain by following the NeOn method for Ontology Engineering proposed by Suárez-Figueroa et al. (2012). This was chosen because it provides a simple and flexible set of recommendations for a number of scenarios which are similar to the scenario at hand.

Within the NeOn method, I have chosen a *waterfall* life cycle. Unlike iterative or incremental life cycles, the waterfall approach aims to generate a useful formalisation of a domain following a single and linear sequence of phases. This life cycle is recommended by the NeOn method for ontologies which cover a *small* and *well-understood* domain. Although there is no exact definition of the *small* and *well-understood* criteria, I argue that the core concepts behind step-by-step instructions and to-do checklists are few and simple, thus fulfilling the *small* criterion. Moreover, this is a domain that a large amount of non-expert web users use on a daily basis. Therefore, it can be said to be a *well-understood* domain.

In the NeOn method, the decision as to which phases to conduct during the waterfall approach depend on the context in which the formalisation is to be developed. In the context of this thesis, it can be observed that there exists an abundance of both unstructured resources, such as instructional websites, and structured resources, such as process description languages. This situation relates to two of the nine scenarios described in NeOn, namely “Scenario 2: Reusing and re-engineering non-ontological resources” and “Scenario 4: Reusing and re-engineering ontological resources” (Suárez-Figueroa et al. 2012). These scenarios correspond to a six-phase waterfall life cycle, composed of the following phases (1) initiation, (2) reuse of ontological and non-ontological resources, (3) re-engineering of ontological and non-ontological resources, (4) design phase, (5) implementation phase and (6) maintenance phase.

The method presented in this thesis follows this six-phase waterfall life cycle, with the exception of phases 2 and 3. In the NeOn method, these two phases differ in their focus, which is re-use for the second, and re-engineering for the third. The method adopted in this thesis instead partitions the content of these phases in a different way, focusing on non-ontological resources in phase 2, and ontological resources in phase 3. The proposed method, composed of the following six phases, will now be described.

First Phase The first phase of this cycle is the *initiation* phase. The main goal of this phase is the definition of a set of requirements that will guide the formalisation of the domain. These primarily consist of a set of minimal functional requirements that the formalisation should fulfil.

Second Phase The second phase is the *reuse and re-engineering of non-ontological resources* phase. During this phase existing non-ontological resources will be considered for inclusion in the formalisation of the domain. If such resources cannot be directly used in the formalisation of the domain, the option of adapting them, or “re-engineering” them to suit the needs of the domain will be considered. This phase will focus on collecting relevant resources and “reverse-engineering” their underlying structure and components.

Third Phase The third phase is the *reuse and re-engineering of ontological resources* phase. This phase can be considered equivalent to the previous phase with the exception of considering ontological resources instead of non-ontological ones. This phase will focus on searching for and comparing ontological resources, as well as deciding at which level of abstraction they can be reused or re-engineered.

Fourth Phase During fourth phase, called the *design* phase, the terminology and patterns discovered in the previous two phases are used to formalise a data model, which defines the type of information the formalism will represent. This model does not need to be directly computable.

Fifth Phase The fifth phase, called the *implementation* phases, involves the materialisation of the model defined in the previous phase into a concrete representation of the model using a computable data format.

Sixth Phase The six phase, called the *maintenance phase*, is triggered in case errors are discovered in the formalisation, or if new versions of the formalisation need to be created. Updates and error-fixes are common to most ontology life cycles, just as bugs are pervasive in computer programming. However the details of such procedures are not relevant to this thesis and therefore this phase will not be described in detail.

4.4 First Phase: the Definition of the Knowledge Requirements

The first phase of the NeOn method focuses on the definition of a set of knowledge requirements. Knowledge bases are a type of database, and the main purpose of a database is to provide information by answering queries. It is therefore natural to define the minimum functionalities required by a knowledge base in terms of queries. To do so, I define a set of *competency questions*, namely the set of queries that the knowledge base should be able to answer.

In order to define these competency questions, I will consider two core applications of a knowledge representation formalism that models instructional knowledge. The first one is the use of this knowledge to discover information on how a task can be completed. Intuitively, this is the typical use of a set of instructions. The second application is the use of this knowledge to track the execution of a task using a checklist approach.

In this thesis it is assumed that when a user is reading a set of step-by-step instructions to complete a specific task, he or she is interested in answering at least one of the following two questions:

Q1 What requirements must be satisfied before performing the task? In other words, how can the user determine whether the task can be performed? Typical answers to this questions include the ingredients required by a recipe, or more abstract conditions, such as the requirement "it must be night-time" for a set of instructions on "how to stargaze".

Q2 How can the task be completed? Or in other words, how can the task be decomposed into simpler tasks? A typical answer to this question would retrieve the list of steps of the set of instructions.

If we consider the second application, we can now imagine a user accessing a to-do checklist to complete a set of instructions. I will assume that such user is interested in answering at least one of the following six questions.

Q3 Which tasks are in the checklist? A user might be simply interested in viewing the various elements of the checklist.

Q4 Which tasks have been completed in the checklist? In other words, which tasks have been already checked off?

Q5 Which tasks are ready to be completed in the checklist? Checklists can enforce the ordering in which tasks should be completed. This ordering can determine which tasks can or cannot be completed at any given moment. It should be noted that not all tasks which *can* be completed *should* to be completed. Tasks which should be completed are the focus of question `Q7` instead.

Q6 Which tasks are being completed by following a separate checklist? Complex sets of instructions might be decomposed into more than one checklist in order to provide a more structured view of all the tasks that needs to be completed. A user looking at a checklist might want to discover which of its elements are decomposed by other checklists.

Q7 Which tasks can be completed at the current time to advance the progress towards completion of the checklist? Arguably, this question captures the main purpose of a checklist, namely to be able to determine what is the next step that should be performed.

Q8 How can a checklist to complete a task be generated from a set of instructions? If a checklist does not already exist, a user might be interested in creating one based on an existing set of instructions.

A more formal definition of these queries will be presented in Section 4.7.3. These competency questions, which are here described in natural language, will be formally defined by taking into account the terminology of the chosen knowledge representation model.

4.5 Second Phase: Reuse and Re-Engineering of Non-Ontological Resources

The second phase of the NeOn method focuses on the reuse or re-engineering of relevant non-ontological resources, and in particular instructional web pages. This phase addresses the question of what kind of knowledge can be extracted from instructional web pages. The goal of this analysis is to identify the most important components, such as “steps”, that are used to represent step-by-step instructions. This analysis is not concerned with domain dependent aspects, such as “cooking time” or “computer requirements”, which are common in certain domain, but uncommon in others.

Following the methodology described in Section 3.1.2, I have adopted a number of search methods and selection criteria to identify relevant instructional websites to analyse. The selection criteria used to determine whether a website should be considered are the following:

- The website contains textual instructions in English.
- The website is domain-independent. More specifically, it should contain sets of instructions from different domains, such as the cooking, technology and DIY domains.
- The website is *large*. I will consider websites with over 10K sets of instructions to be large.

Other criteria that could be used to estimate the importance of a website are factors such as the average number of active users and content creators. However, due to the difficulty in obtaining these user statistics for all the websites, I have chosen not to consider them.

Having defined these selection criteria, I have applied two search methods to discover relevant instructional websites. The first method is based on a keyword-based search using web search engines. The second method searched for mentions of instructional websites in online reviews. These reviews were discovered using keyword-based

Website	Number of articles	Tasks	Steps	Step ordering	Requirements	Methods
eHow[3]	>3M (2011)	○	○	○	○	✖
Instructables[4]	>245K (2016)	✓	○	○	○	✖
wikiHow[5]	>195K (2016)	✓	✓	✓	○	○
WonderHowTo[6]	>100K (2008)	○	○	○	○	✖
Snapguide[7]	>44K (2014)	✓	✓	✓	○	✖
Howcast[8]	>30K (2016)	✓	✓	✓	○	✖

Table 4.1: The occurrence of instructional concepts across some of the main instructional websites indicated as "always" (✓) "sometimes" (○) and "non-present" (✖). The number of published articles in English and the year of this measurement is also given.

search using web search engines. This search method can also be considered as a criteria to select prominent websites. Prominent websites, in fact, are more likely to feature in online reviews and to rank higher in search engine results.

I have conducted this search using the Yahoo[1] and Google[2] search engines, considering only the first two pages of results. When applying the first method, I have used keywords such as: "how to", "step by step" and "instructions". When applying the second method, I have searched for reviews using keywords such as: "instructional websites", 'how to websites", "best" and "review".

On the basis of this search, I have identified six websites that match the selection criteria. These websites are listed in Table 4.1, along with information on the number of articles that they contain. After identifying these websites, I have analysed several sets of instructions for each website to discover the main components that these websites have in common. The occurrence of these components in each website is also detailed in Table 4.1.

The most important component, which can be found in all step-by-step instructions, is the notion of what can be expected to be accomplished after following the set of instructions. In my instructional model this is called a *task*. More specifically, this term is used to refer to things which can be accomplished, or in other words "checked-off" from a to-do list. Under this definition, the term task can be used not only to

[1] `https://yahoo.com/` (accessed on 17/10/2017)
[2] `http://www.google.com/` (accessed on 17/10/2017)
[3] `http://www.ehow.com/` (accessed on 17/10/2017)
[4] `http://www.instructables.com/` (accessed on 17/10/2017)
[5] `http://www.wikihow.com/` (accessed on 17/10/2017)
[6] `http://www.wonderhowto.com/` (accessed on 17/10/2017)
[7] `http://snapguide.com/` (accessed on 17/10/2017)
[8] `http://www.howcast.com/` (accessed on 17/10/2017)

refer to the main goal of a set of instructions, but also to its sub-tasks, and even to the process of satisfying certain preconditions, such as obtaining the required tools and ingredients.

Requirements such as "1 liter of milk", in fact, are interpreted in the context of the execution of task as the process of "obtaining" such a requirement. In a to-do list, the ingredient "1 liter of milk" can be checked off after it is bought at a store, in a similar way as the step "boil the milk" can be checked off after it is performed. Therefore the distinction between actions and objects, which is fundamental to most artificial process formalisations, becomes blurred in the domain of human-written instructions. A cooking recipe, for example, could mention the ingredient "milk" as an input, or it could mention "get milk" 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, this concept of task diverges from typical process formalisations in that it does not enforce a distinction between actions and objects. If this distinction is required, it can be easily implemented as a specification of the concept of task, for example into the sub-concepts of "action task" and "requirement task".

The second most important component, which can be found in virtually all step-by-step instructions, is the notion of a *step*. The main task that a set of instructions describes can be decomposed into a set of steps such that the completion of those steps is equivalent to the completion of the main task.

Another common component is the notion of *dependency* between tasks. A task depends on, or *requires*, another task, when the former should be executed before the completion of the latter. A common type of dependency can be inferred by the ordering between steps, where the possibility of completing one step can depend on the condition that the previous step has already been accomplished. Given that tasks can also represent the requirements of a set of instructions, the notion of dependency also captures the notion of the requirements of a task, such as a list of ingredients. The task of gathering such ingredients is a requirement of the steps that make use of those ingredients.

The last core component found across instructional websites is the concept of *methods*. Across all domains of know-how one can observe that for every task that one might want to accomplish, multiple approaches to solve it are possible. This is true even if an optimal set of instructions to accomplish a task is known. In practice, it is very common to find similar sets of instructions that can be followed to achieve the same goal. For example, multiple pancake recipes can be found on the web. Although they might lead to the creation of different types and flavours of pancake, they can all be said to achieve the generic goal of "creating a pancake". These instructions can therefore be said to be different methods to achieve this goal. On the website wikiHow, different sets of instructions to achieve the same goal are explicitly called "methods".

4.6 Third Phase: Reuse and Re-Engineering of Ontological Resources

The third phase of this method focuses on the reuse or re-engineering of relevant ontological resources and, more specifically, process representation languages. To do so, I follow the methodology described in Section 3.1.2. This methodology involves identifying the most relevant fields where process representation languages are used, and then analysing the most prominent approaches of each field. These approaches will be analysed to incorporate as far as possible the best practices, terminology and knowledge patterns that have proven successful in the past.

The body of work on formalising processes is extensive and it comes from several fields. A review of the most related ones has been presented in Section 2.2. From this review, five main areas can be identified, namely Automated Planning (AP), Web Services (WS), Semantic Web (SW), Social Computation (SC) and representations of human instructions (HI). The AP and WS fields have been chosen because of their strong focus on the representation and automation of processes. The SW field has been chosen because it aims to provide reusable semantic representations of knowledge, including procedural knowledge. The SC field has been chosen because it investigates processes and interactions between multiple humans and machines. Finally, I have collected under the HI area a number of approaches that extract information about human instructions from textual documents. Although HI approaches primarily focus on knowledge extraction, they also provide tangible examples of how such knowledge can be represented.

Having identified a set of relevant fields, I have searched for the most relevant approaches in these fields using Scopus[9], Web of Science[10] and Google Scholar[11]

[9]`https://www.scopus.com/` (accessed on 17/10/2017)

[10]`http://ipscience.thomsonreuters.com/product/web-of-science/` (accessed on 17/10/2017)

[11]`http://scholar.google.com/` (accessed on 17/10/2017)

Area	Name	Publication	Citations		
			Web of Science	Scopus	Google Scholar
AP	PSL	Grüninger and Menzel (2003)	79	109	187
AP	PDDL	Mcdermott et al. (1998)	-	-	1382
AP	HTN	Erol et al. (1994b)	104	271	778
WS	OWL-S	Martin et al. (2007)	187	318	778
SW	Schema.org	Guha et al. (2016)	3	12	35
SW	Activity Streams	Guy et al. (2011)	-	26	43
SC	CompFlow	Luz et al. (2014)	2	2	4
SC	CrowdLang	Minder and Bernstein (2012)	14	24	46
HI	Recipe Model	Kiddon et al. (2015)	-	9	10
HI	Situation Ontology	Jung et al. (2010)	11	28	52
HI	Medical Ontology	Song et al. (2011)	-	13	24

Table 4.2: Existing process formalisation approaches that are highly relevant to the knowledge representation problem discussed in this chapter. The area each formalism belongs to is listed under the column *Area*: Automated Planning (AP), Web Services (WS), Semantic Web (SW), Social Computation (SC) and human instructions (HI). The name of the approach and its main related publication are listed, respectively, under columns *Name* and *Publication*. The remaining columns provide the citation count of the papers given by three academic search engines, or the symbol "-" if the paper could not be found. The citation count was updated on September 2017.

academic databases and search engines. This search was based on keyword searches and it made use of existing literature reviews of the different fields.

The approaches I have selected, and the field they belong to, are listed in Table 4.2. This table also details the main academic publication of each approach along with its citation counts provided by the three academic search engines. These citation counts are provided only as a partial indication of the impact of the approaches, and I have taken into account other factors when evaluating their overall importance and relevance.

Being mature fields, it is not surprising that AP and WS approaches generated high impact in terms of citations. In the AP field I have chosen PDDL, HTN and PSL not only because they are well established approaches, but also because of their qualitative differences. In particular, HTN is a major alternative planning paradigm to classical planning, of which PDDL is a prime example. PSL on the other hand, is an international standard more focussed on the representation of business processes and workflows. In the WS field, many formalisms have been developed to represent Web

Services. Amongst those I have chosen to analyse OWL-S because it is a well established and because it aims to provide reusable semantic representations of services.

In the SW area, Schema.org was chosen as it is the major industry-led standard to semantically enrich web resources. The work on Activity Streams was chosen, instead, because it is an official recommendation of the W3C consortium.

My research did not find well established standards in the SC and HI areas. For this reason, I have chosen to review a number of approaches that, despite not being widely adopted yet, are highly relevant to the work presented in this thesis. In particular, the approaches I have chosen in the SC field describe formal representations of processes involving multiple humans and machines. The chosen approaches in the HI field, instead, describe formal representations of human instructions extracted from natural language texts.

Having identified a number of relevant ontological approaches to represent procedural knowledge, I proceeded to analyse them to identify semantically similar concepts which are common across these approaches. Table 4.3 lists the most relevant concepts that I have identified, along with the terminology used by each approach to refer to that concept, if present.

Similarly to the analysis on the non-ontological resources, it is not surprising that the concept of a task (also called "action", "activity", "process", "method" or "direction") appears in virtually all of the formalisations. The concept of the execution of a task (also called "activity occurrence", "activity", or "action"), although common, is not found in all of the languages. This is partially the result of several of those languages describing processes, but not the execution of these processes. The concept of an environment, namely the context in which activities are executed, is often not explicitly described. This is a result of the fact that languages that describe the execution of tasks might assume a single environment where the execution takes place. For example, the solution to a PDDL problem is a list of instantiated actions. This list can be interpreted as an environment, and each item in the list as an execution. However, PDDL does not have an explicit representation of these concepts.

In most of these formalisms, tasks can only be executed when their dependencies are satisfied. These dependencies can be called "preconditions", "truth constraints", "inputs", the constraints of a "sequence flow", "connections" or "sequence relations". The abstract notion of dependency is often specialised in one of three variants (1) *inputs*, when a task requires the availability of an object, (2) *preconditions*, when a task requires a proposition to be true, and (3) *ordering*, when a task requires another

Name	Execution	Environment	Task	Dependency	Effects	Method	Decomposition	Var. Binding	Primitive Activity	Object	Location	Time	Agents
PSL Outer-Core	Act. Occurrence	✖	Activity	✖	✖	✖	Subactivites	✖	Primitive Activity	Object	✖	Timepoint	✖
PDDL	✖	✖	Action	Precondition	Effect	✖	✖	✖	✖	Object	✖	✖	✖
HTN	✖	✖	Task	Truth Constraints	Postconditions	Method	Expansion	✖	Primitive Task	✖	✖	✖	✖
OWL-S	✖	✖	Process	Input, Precondition	Output, Results	✖	Composite Process	Binding	Atomic Process	✖	✖	✖	Participant
Schema.org	Action	✖	HowToDirection	✖	Result	✖	HowToStep	✖	✖	HowToItem	Location	Duration	Agent
Activity Streams	✖	✖	Activity	✖	✖	✖	✖	✖	✖	Object	✖	✖	Actor
CompFlow	Activity	Workflow	Task	Input	Output	✖	✖	✖	✖	✖	✖	✖	Worker
CrowdLang	✖	✖	Task	Sequence Flow	✖	✖	Divide-and-Conquer	✖	✖	✖	✖	✖	✖
Recipe Model	✖	✖	Action	Connection	✖	✖	✖	✖	✖	Ingredients	Location	✖	✖
Situation Ontology	✖	✖	Action	✖	✖	Goal	✖	✖	✖	Object	Location	✖	✖
Medical Ontology	Action	✖	Method	Sequence Relation	✖	✖	✖	✖	✖	Target	✖	Time	✖

Table 4.3: Features of the formalisms listed in Table 4.2 used to represent processes and their executions. The name of the formalisms is listed under the column *Name*. The remaining columns represent different semantic concepts. The cells at the intersection of a formalism and a semantic concept contain the symbol ✖ if the concept is not expressed in the formalism, or if it is, the label associated with that concept in that representation.

task to have already been executed.

The concept of *effects* is quite common across procedural languages, and it describes which assumptions can be made after an action has been executed. It is usually specialised into two types: (1) postconditions, which refer to the propositions which can be assumed to be true after the task has been executed, and (2) outputs, namely the objects that can be assumed to be available after the task has been executed. The concept of *method* describes one or multiple ways to complete a task. A method can be interpreted as which task can be assumed to be complete after another one is accomplished. In this last case, a method can be seen as a type of effect. It should be mentioned that the term "postcondition" can have a very different interpretation in some other contexts, as it could refer to the set of conditions that a planner needs to guarantee to be true after a task has been executed.

Another common concept is the concept of a decomposition (also related to the terms "step", "composite-process", "divide-and-conquer", "subactivity" or "expansion"). This concept expresses the fact that the execution of a set of tasks (or sub-tasks) is one possible way to complete another task (or super-task).

The decomposition of one task into a different set of tasks is usually motivated by the need to decompose a complex task into a set of *primitive activities*. A primitive activity is usually defined as an activity that can be completed directly without requiring further refinement or decomposition.

The concept of variable bindings is common within the web-service community to indicate how the inputs and the outputs of different services can be linked to each other to compose the services together.

Entities which are not tasks are sometimes classified as *agents*, *objects*, *locations* and *times*. Agents (also called "actors", "participants" or "workers") represent the entities which are responsible for completing activities. Objects (also called "items", "ingredients" or "targets"), on the other hand, typically represent things which do not have agency. The notion of location and time (also related to the terms "duration" and "timepoint") are also common in domains where the actions need to be timed carefully and performed in specific locations. Although widespread, the use of the terms agents, objects, locations and times is typically context dependent. For example, the term agent might not need to be specified in a scenario where only one agent is performing a task, and the term object might not be needed to perform activities which do not require the manipulation of specific objects.

Base Predicates	Short Description
task(x)	x is a task
execution(i)	i is an execution of a task
environment(e)	e is an environment
has_step(x,y)	y is a step of x
has_method(x,y)	y is a method of x
requires(x,y)	x requires y
requires_one(x,y)	y is a sufficient requirement of x
has_task(i,x)	i is an execution of task x
has_env(i,e)	i is an execution in environment e
has_goal(e,x)	environment e has the goal to complete x
success(i)	execution i has succeeded
sub_env(e,a)	e is a sub-environment of a
has_value(i,v)	execution i resulted in value v
has_constant(x,v)	task x is associated with constant v
binds(x,y)	if y is complete in an environment, so is x in all its related environments
Derived Predicates	**Short Description**
finished(e)	environment e has achieved its goal
active(e)	executions in environment e can become ready
complete(x,e)	task x has been completed in environment e
ready(x,e)	task x is ready to be executed in environment e
related(e,a)	environments e and a are related
value(v,x,e)	v is the value of task x in environment e

Table 4.4: The logical predicates of the PROHOW model along with a short description of their meaning.

4.7 Fourth Phase: Data Model

The fourth phase of this method involves defining a data model based on the on the terminology identified in the previous two phases. This terminology points at the following four main concepts used to describe instructions: (1) tasks, (2) steps, (3) methods and (4) requirements. The execution of a task can be represented with the concepts of: (1) environments, (2) executions, (3) the goal of the environments and (4) the status of completion of an execution. Having already discussed the relevance and meaning of these terms, this section combines this terminology with the knowledge requirements acquired in the previous phases to formalise the PROcedural know-HOW (PROHOW) data model using First Order Logic (FOL). Sufficient and necessary defini-

Class	Variable Names
Task	$x\ y\ z$
Execution	$i\ u\ j$
Environment	$e\ a\ o$
Thing	v

Table 4.5: Variable naming convention adopted in this chapter. The variables listed in this table can be assumed to refer to entities of the given class.

tions of terms will be denoted by the $\iff$ symbol, which represents a logical biconditional connective. Sufficient definitions will be denoted by the backward implication symbol $\Longleftarrow$, which is equivalent to the FOL implication connective if the antecedent and consequent are exchanged. The predicates used in this model are listed in Table 4.4. The *base predicates* are those that need to be explicitly asserted to be considered true, while the *derived predicates* can remain implicit as they can be derived from the base predicates. As an aid to the reader, the variables used in the logical axioms will follow the convention shown in Table 4.5.

Figure 4.2 illustrates the predicates of the PROHOW model with respect to the unary predicates *task*, *execution* and *environment*, which represent the three main types, or classes, of this model. A formal definition of the relations between predicate and classes will be given in Axiom 23. In this figure, classes are represented as rectangles. Binary predicates are represented by an arrow from the class of the first variable to the class of the second variable. The unary predicates *success*, *finished* and *active* are represented as rounded rectangles linked to the class of its variable. The ternary predicate *value* is represented by two dashed connectors ending in a round circle and an arrow, starting from the class of its first variable and ending, respectively, at the classes of its second and third variables.

4.7.1 Working Example

To better explain the PROHOW model this section introduces two sets of instructions as a working example. These instructions are represented as HTML web pages.[12] Figures 4.3 and 4.4 display screenshots of these web pages as they are displayed by a web browser. For the sake of simplicity, these examples have been created to be very

[12]These web pages are available here: `http://paolopareti.uk//r/sampleinstructions.htm` (accessed on 17/10/2017)

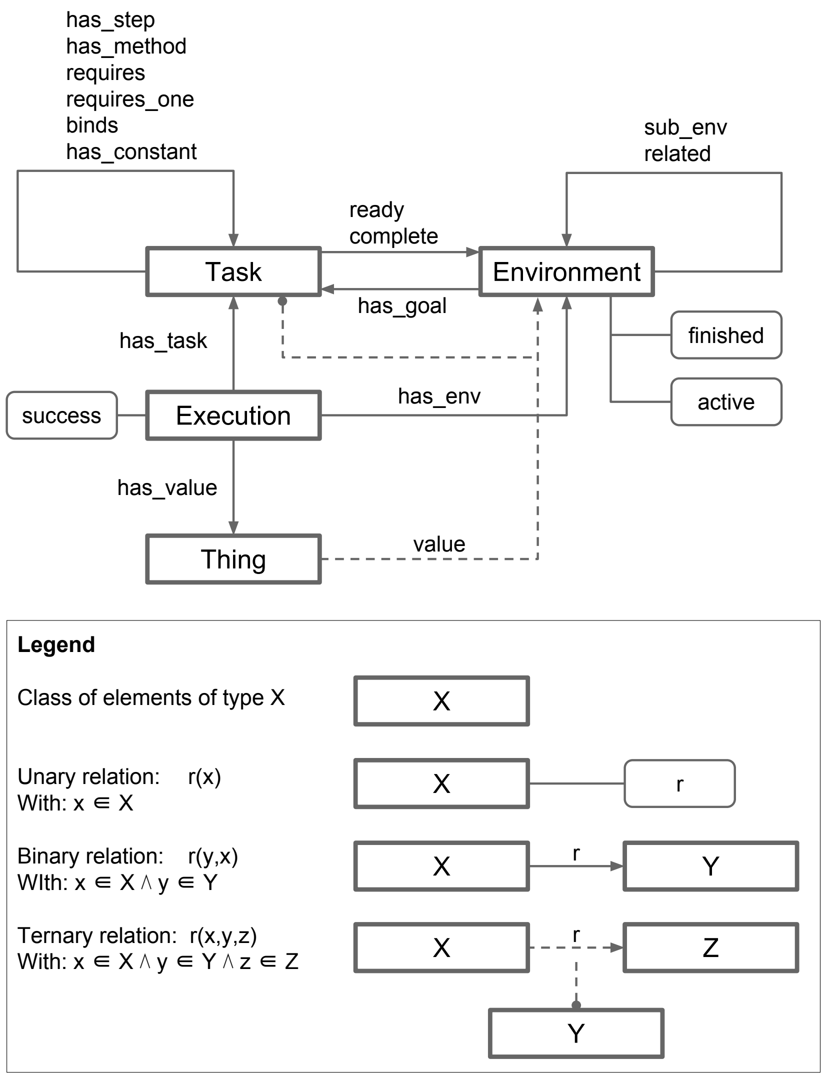
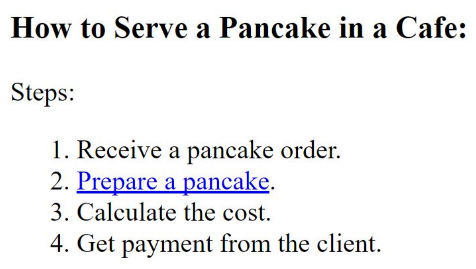


Figure 4.2: A model of the PROHOW predicates as a set of classes, relations and properties representing the axioms of Axiom 23.

concise.

In this example, the manager of a café has decided to publish a number of sets of instructions online for the employees of the café to follow. Figure 4.3 represents one such set of instructions, which explains how to serve pancakes in the café. This set of instructions only includes four steps. First the employee needs to receive a pancake order. Then, he or she will start preparing the pancake. After that, the cost for the pancake will be calculated, and then the payment will be made by the client. The manager has linked the second step to another web page which explains how the pancake should be prepared.

This second web page, displayed in Figure 4.4, details a number of ingredients and

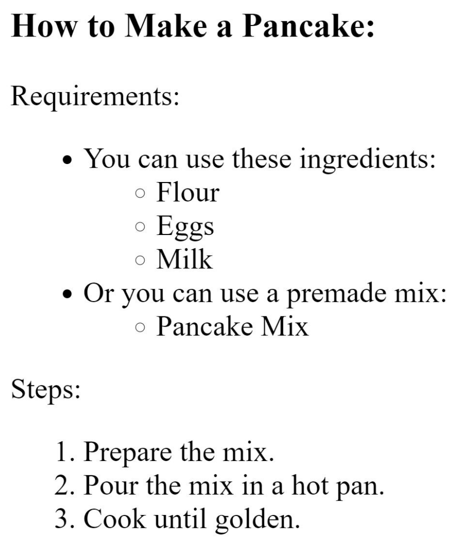


How to Serve a Pancake in a Cafe:

Steps:

1. Receive a pancake order.
2. Prepare a pancake.
3. Calculate the cost.
4. Get payment from the client.

Figure 4.3: A screenshot of the example web page containing restaurant instructions. The step “Prepare a Pancake” contains a link to the set of instructions in Figure 4.4



How to Make a Pancake:

Requirements:

- You can use these ingredients:
  - Flour
  - Eggs
  - Milk
- Or you can use a premade mix:
  - Pancake Mix

Steps:

1. Prepare the mix.
2. Pour the mix in a hot pan.
3. Cook until golden.

Figure 4.4: A screenshot of the example web page containing a pancake recipe.

the steps to follow in order to prepare the pancake. To complete this set of instructions, the employees first need to gather the necessary requirements. They have a choice of using a pre-made pancake mix, or alternatively to use eggs, milk and flour. They can then follow the steps to cook the pancake. It should be noted that whenever steps are presented as a numbered list, it will be assumed that they need to be performed in the same order as they appear in the list.

Additionally to these web pages, the manager provides the employees with an interface where they can can log their progress as they follow instructions using a checklist approach. For example, whenever an employee is preparing a pancake, he or she will check the boxes of the ingredients that are being used and the box of each step once it is complete. An example of what these checklists could look like is presented in Figure 4.5. The checklist on the left refers to the set of instructions on how serve pancakes in the café. An employee has checked the box relative to the first step, thus indicating that a pancake order has been received. In order to prepare the pancake, the employee has

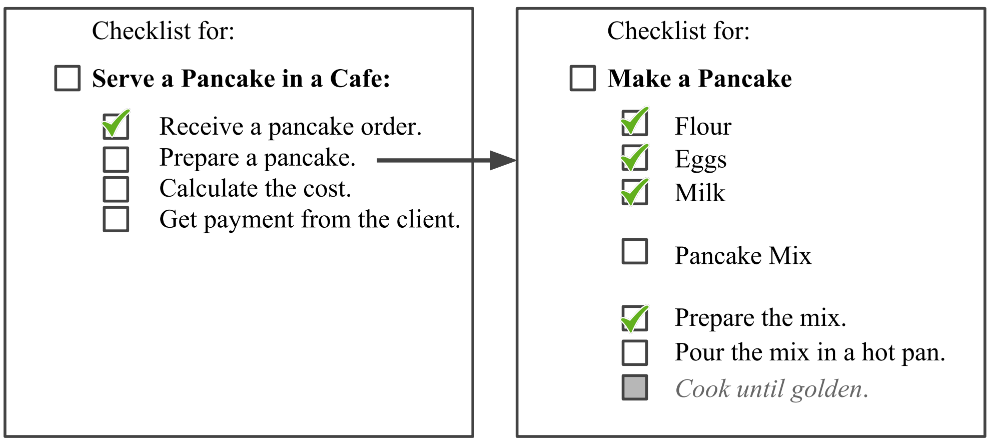


Figure 4.5: A possible visualisation of checklists for the sets of instructions in figures 4.3 and 4.4.

then started to follow the checklist on the right. The relation between the two checklists is indicated by the arrow from the second step of the checklist on the left to the checklist on the right. In this last checklist, the boxes related to the ingredients eggs, milk and flour have been checked, meaning that those ingredients have been selected instead of the pre-made pancake mix. The first step of the instructions, “Prepare the mix”, has been checked off. This represents the fact that the employee has already performed that step. To continue following the instructions, the employee can now perform the second step. The third step, namely “Cook until golden”, is greyed out to indicate that it cannot be checked off yet. This is a result of the constraint that the second step needs to be completed before the third one.

The manager hopes that these checklists will help the employees follow the instructions thoroughly. Moreover, by logging the data from these checklists, the manager will be able to collect information on which ingredients are being used; which procedures are being followed; and how long each step usually takes. In the next section I will present the axioms of the PROHOW model, and explain how they can be used to model these sets of instructions and their execution using a checklist.

4.7.2 Axioms of the Model

In this model I will use the terms *entities* and *things* interchangeably and according to the OWL interpretation[13] of a thing, namely as the most generic concept. In other words, everything can be said to be a thing. An interpretation of the FOL variables

[13] `https://www.w3.org/2002/07/owl#Thing` (accessed on 17/10/2017)

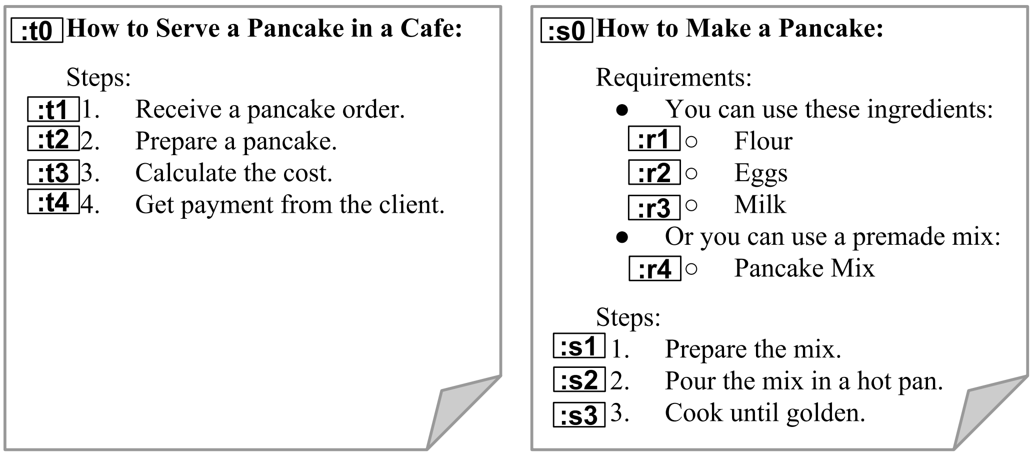

:t0 How to Serve a Pancake in a Cafe:
Steps:
:t1 1. Receive a pancake order.
:t2 2. Prepare a pancake.
:t3 3. Calculate the cost.
:t4 4. Get payment from the client.
:s0 How to Make a Pancake:
Requirements:
You can use these ingredients:
:r1 Flour
:r2 Eggs
:r3 Milk
Or you can use a premade mix:
:r4 Pancake Mix
Steps:
:s1 1. Prepare the mix.
:s2 2. Pour the mix in a hot pan.
:s3 3. Cook until golden.

Figure 4.6: A schematic representation of the sets of instructions displayed in figures 4.4 and 4.4 with additional URI identifiers attached to the main entities which they are made of.

used in this model is a mapping between each variable and an entity. A named set of entities is a *class*. In the context of an ontology, a class can also be called a *concept*. Entities that are included in such a set are said to *belong* to that class, or of being of that *type*. In this FOL model, I will define class membership using unary predicates.

Concrete examples of entities are presented in Figure 4.6. This figure contains a schematised representation of the sets of instructions displayed in figures 4.4 and 4.4, with the addition of URI identifiers for a number of entities contained in these instructions. To enhance readability, I name the URIs of my examples using the empty prefix ":" as a shorthand for a full namespace, such as the one defined in Table 2.1. For example, the URI `:t0` refers to the set of instructions on "How to Server a Pancake in a Café"; URI `:s1` refers to the step "Prepare the mix"; and URI `:r1` refers to the requirement "Flour".

With respect to a checklists, other examples of entities can be observed in Figure 4.7. The two checklists are identified, respectively, by URIs `:e0` and `:e1`. Each check-box is also identified with a URI. For example, URI `:u5` refers to the check-box for the step "Prepare the mix" contained in checklist `:e1`.

The type of entities central to this model is the class of *tasks*. The fact that entity x is a task is defined by the predicate $task(x)$ (Definition 4.7.2). In PROHOW the interpretation of a task is quite generic and, as described in Definition 4.7.1, it is made of all the things that can be checked off from a to-do list. For example, the ingredient "Flour" (entity `:r1`) and the step "Prepare the mix" (entity `:s1`) from our working example are

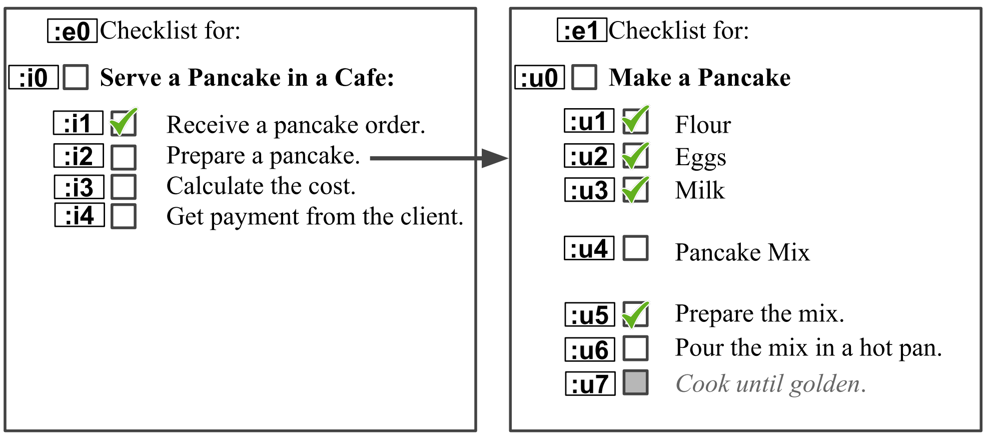


Figure 4.7: A representation of the checklists of Figure 4.5 with additional URI identifiers attached to the main entities which they are made of.

tasks. These facts are represented by the statements *task*(`:r1`) and *task*(`:s1`). Tasks can be seen as things that can be acted upon. Acting upon a task might involve performing a specific action, such as in the case of the step "Prepare the mix". Acting upon a task might also involve retrieving or verifying the availability of something. For example, an employee of the café might look for the ingredient "flour", or verify that it is already available in the kitchen, before checking its box in the to-do list.

Definition 4.7.1. *Task* A task is something that can be acted upon to be checked off a to-do list.

Definition 4.7.2. *task*(x) x is an entity of type *Task*.

Another core concept of this model is the checklist where instructions are executed. Checklists can be seen as providing a context, or *environment*, to the execution of instructions. For example, we can imagine two different customers ordering pancakes at the café, which are then prepared by two different employees. If one employee finishes preparing a pancake before the other, the set of instructions on "How to Make a Pancake" might be complete in one checklist, but it might still be a work in progress in the other. In the PROHOW model, checklists correspond to the concept of *environment* (Definition 4.7.3). The predicate *environment*(e) represents the fact that an entity e is an environment (Definition 4.7.4). In Figure 4.7 two examples of environments are provided, namely `:e0` and `:e1`. This is represented by the statements *environment*(`:e0`) and *environment*(`:e1`).

Definition 4.7.3. *Environment* An environment represents a specific intention to achieve a certain goal task. An environment corresponds to a to-do checklist.

Definition 4.7.4. *environment*(e) Entity *e* is an *Environment*.

The task that an en environment was created to accomplish is its *goal*. The predicate *has_goal*(e,x) (Definition 4.7.5) is used to indicate that the main task (or goal) of environment *e* is task *x*. For example, after a pancake order is placed, an employee might start a new checklist to "prepare a pancake". In this case, the task "prepare a pancake" can be said to be the goal of that checklist. In the example checklists provided in Figure 4.7, for example, the goal of checklist `:e0` is accomplishing the task `:t0`: "Serve a Pancake in a Café" (*has_goal*(`:e0`, `:t0`)).

The goal of checklist `:e1`, instead, is more complex to determine. In our example, checklist `:e1` is not independent from checklist `:e0`. More specifically, the employees of the café follow the set of instructions on "How to Make a Pancake" (`:s0`) in order to complete the step "Prepare a pancake" (`:t2`) of the other set of instructions. In this case, checklist `:e1` is considered as a nested checklist of checklist `:e0`. In order to capture the reason why the nested checklist is being followed, the goal of checklist `:e1` is set as `:t2` (*has_goal*(`:e1`, `:t2`)).

Definition 4.7.5. *has_goal*(e,x) The goal of *Environment e* is achieving task *x*.

Each environment has exactly one goal. In other words, each checklist is meant to achieve something. This statement is represented as a consequence of axioms 1 and 2.

Axiom 1. Each environment has a goal.

$$\text{environment}(e) \rightarrow \exists x.\text{has\_goal}(e,x)$$

Axiom 2. Each environment has at most one goal.

$$\neg\exists e,x,y.x \neq y \wedge \text{has\_goal}(e,x) \wedge \text{has\_goal}(e,y)$$

The two environments in our example are related to each other. More specifically, `:e1` is a *sub-environment* of `:e0` because its purpose is to help accomplishing a task in `:e0`, namely `:t2`. We can denote the fact than an environment *e* is a sub-environment of another environment *a*, and consequently that *a* is a super-environment of environment *e*, with the predicate *sub_env*(e,a) (Definition 4.7.6). There is no restriction on the number of sub-environments that an environment can have.

Definition 4.7.6. *sub_env*(e,a) Environment *e* is a sub-environment of environment *a*. The purpose of environment *e* is to accomplish a task in environment *a*.

If we interpret the *sub_env* predicate as a relation between environments, such relation is not meant to be circular. In other words, by following a chain of *sub_env* relations, it should not be possible to visit the same environment more than once. Axiom 3 prohibits such circular relationships making use of the helper predicate *descendent_env*, which corresponds to the transitive version of the *sub_env* relation.

Axiom 3. The sub-environment relation cannot be circular.

$$\begin{aligned}
&\text{descendent\_env}(e,a) \Longleftarrow \text{sub\_env}(e,a)\\
&\text{descendent\_env}(e,a) \Longleftarrow \text{sub\_env}(o,a) \wedge \text{descendent\_env}(e,o)\\
&\neg\exists e.\text{descendent\_env}(e,e)
\end{aligned}$$

Environments that are connected with the sub-environment relation are said to be *related* environments. Predicate *related*(e,a) states that environments e and a are related (Definition 4.7.7). This predicate is defined logically by Axiom 4. In our example, environments `:e0` and `:e1` are related (*related*(`:e0`, `:e1`) and *related*(`:e1`, `:e0`)).

Definition 4.7.7. *related*(e,a) Environment e belongs to the same network of interconnected environments as environment a. The logical definition of this predicate is given in Axiom 4.

Axiom 4. The logical definition of the *related* predicate.

$$\begin{aligned}
&\text{related}(e,a) \iff e = a \vee \text{sub\_env}(e,a) \vee \text{sub\_env}(a,e) \vee\\
&\exists o.\text{related}(e,o) \wedge \text{related}(o,a)
\end{aligned}$$

In order to achieve the goal of an environment, a number of tasks need to be executed. For example, in order to achieve the goal of preparing a pancake, an employee might have to prepare the pancake mix. In a checklist, specific executions of tasks are represented as check-boxes. In the PROHOW model check-boxes are represented with the concept of an *execution* (Definition 4.7.8). The predicate *execution*(i) represents the fact that an entity i is an execution (Definition 4.7.9). For example, the check-box `:u1` in Figure 4.7 is an execution (*execution*(`:u1`)).

Definition 4.7.8. *Execution* An execution represents a specific attempt to perform a certain task. It corresponds to a check-box in a to-do checklist.

Definition 4.7.9. *execution*(i) Entity i is an *Execution*.

As each check-box belongs to a checklist, each execution belongs to an environment. Predicate *has_env*(i,e) is used to represent the fact that an execution i belongs to a specific environment e (Definition 4.7.10). The task that an execution is meant to achieve is represented by predicate $\mathit{has\_task}(i,x)$ (Definition 4.7.11). For example, the check-box `:u1` in Figure 4.7 belongs to the checklist `:e1` ($\mathit{has\_env}($`:u1`, `:e1`$)$) and it is meant to achieve task `:r1` ($\mathit{has\_task}($`:u1`, `:r1`$)$).

Definition 4.7.10. $\mathit{has\_env}(i,e)$ Execution i belongs to environment e.

Definition 4.7.11. $\mathit{has\_task}(i,x)$ Execution i is an attempt to achieve task x.

In the PROHOW model, each execution belongs to exactly one environment, and it is meant to achieve exactly one task. These two statements are represented, respectively, as consequences of axioms 5 and 6 and axioms 7 and 8. It should be noted there is no restrictions on how many executions a task can have.

Axiom 5. Each execution has an environment.

$$\text{execution}(i) \rightarrow \exists e.\text{has\_env}(i,e)$$

Axiom 6. Each execution has at most one environment.

$$\neg\exists i,e,a.e \neq a \wedge \text{has\_env}(i,e) \wedge \text{has\_env}(i,a)$$

Axiom 7. Each execution has a task.

$$\text{execution}(i) \rightarrow \exists x.\text{has\_task}(i,x)$$

Axiom 8. Each execution has at most one task.

$$\neg\exists i,x,y.x \neq y \wedge \text{has\_task}(i,x) \wedge \text{has\_task}(i,y)$$

It is important to notice that the three classes of entities defined in this model, namely *Task*, *Environment* and *Execution*, are not overlapping. Axiom 9 defines these three classes as disjoint.

Axiom 9. The classes *Task*, *Environment* and *Execution* are disjoint.

$$\begin{aligned}
&\text{task}(x) \rightarrow \neg\text{execution}(x) \wedge \neg\text{environment}(x)\\
&\text{environment}(e) \rightarrow \neg\text{execution}(e) \wedge \neg\text{task}(e)\\
&\text{execution}(i) \rightarrow \neg\text{task}(i) \wedge \neg\text{environment}(i)
\end{aligned}$$

When the execution of a task is successfully accomplished, the corresponding check-box is checked off from the to-do list. In this model, the predicate $\mathit{success}(i)$ denotes that execution i has been accomplished (Definition 4.7.12). For example, the check-box `:u1` in Figure 4.7 has been checked to indicate that it has been accomplished successfully ($\mathit{success}($`:u1`$)$).

Definition 4.7.12. *success*(i) Execution i is has been successfully accomplished.

A task x can be considered complete in a checklist when its relevant check-box has been checked off. 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. The predicate *complete*(x,e) is used to indicate that task x is complete in environment e. This predicate is defined in Definition 4.7.13. For example, the statement *complete*$(\texttt{:r1},\texttt{:e1})$ indicates that the requirement "Flour" has been obtained in check-list `:e1`. Axiom 10 specifies one way to infer the completion of a task, namely that if an execution i of a task x in an environment e has succeeded, then the task x is complete in that environment.

Definition 4.7.13. *complete*(x,e) Task x has been successfully completed in environment e. The logical definition of this predicate is given in Axiom 21.

Axiom 10. Definition of how a task can be considered complete by the successfull accomplishment of an execution.

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

In a checklist, executions can only be considered as complete or not. It should be noted that this simple model can be easily extended to have other outcomes of an execution i, such as failure, which could be identified, for example, with predicate *failure*(i), disjoint from predicate *success*(i). A failure could be seen as the inability of completing an execution of a task, and therefore suggesting the need to re-instantiate another execution, or trying a different approach which does not involve the failed task.

An environment is said to be finished when its goal is complete. In our example we can observe that, once the pancake has been prepared, there is no reason to continue working on checklist `:e1`. This predicate is defined in Definition 4.7.14 and Axiom 11. In our example, we can observe that if the task `:s0` of completing a pancake is complete in environment `:e1`(*complete*$(\texttt{:s0},\texttt{:e1})$), then that environment is finished (*finished*$(\texttt{:e1})$).

Definition 4.7.14. *finished*(e) The goal of environment e is complete in that environment. The logical definition of this predicate is given in Axiom 11.

Axiom 11. Logical definition of the predicate *finished*.

$$\text{finished}(e) \iff \exists x.\text{has\_goal}(e,x) \wedge \text{complete}(x,e)$$

One of the most prominent feature of step-by-step instructions are the steps. Steps can be seen as a way to simplify a complex task into a set of simpler ones. The relation between a task x and one of its steps y is denoted by the predicate $\mathit{has\_step}(x, y)$ (Definition 4.7.15). In our example, the statement $\mathit{has\_step}(\texttt{:s0}, \texttt{:s1})$ indicates that `:s1`, namely "Prepare the mix" is a step of the task `:s0`, namely "Make a Pancake". Axiom 12 specifies that if a task x has at least one step, and all of its steps are complete in an environment e, then task x is also complete in that environment. Intuitively, this means that a task can be accomplished by accomplishing all of its steps. In our example, task `:s0` can be inferred complete in environment `:e1` if the three steps `:s1`, `:s2` and `:s3` are complete in that environment. Task `:r4`, on the other hand, does not have steps, and therefore cannot be inferred complete using Axiom 12.

Definition 4.7.15. $\mathit{has\_step}(x, y)$ Task y is a step of task x. Completing all the steps of a task is equivalent to completing the task itself.

Axiom 12. Definition of how a task can be completed by completing its steps.

$$\text{complete}(x, e) \Longleftarrow (\exists y.\text{has\_step}(x, y)) \wedge (\forall z.\text{has\_step}(x, z) \rightarrow \text{complete}(z, e))$$

Certain tasks can be completed in multiple ways by following different methods. It is important to notice that in the PROHOW model, whenever a method is followed, a sub-environment needs to be created. This requirement is a consequence of the fact that each task can only be completed in each environment once. In order to allow the same method to be used more than once, each method needs to be executed in its own environment.

The relation between a task x and one its methods y is expressed by the predicate $\mathit{has\_method}(x, y)$ (Definition 4.7.16). This predicate denotes the fact that completing task y is a possible method for completing task x. If a method y of a task x is complete in environment e with goal x, then task y is also complete in environment e (Axiom 13). Intuitively, this means that a task can be completed by completing any of its methods. In our example, if task `:s0` is complete in environment `:e1`, then we can infer that task `:t2` is also complete in that environment, since task `:t2` is the goal of that environment.

Definition 4.7.16. $\mathit{has\_method}(x, y)$ Task y is a method of task x. Completing a method of a task is equivalent to completing the task itself.

Axiom 13. Definition of how a task can be completed by completing one of its methods.

$$\text{complete}(x,e) \Longleftarrow \exists y.\text{has\_goal}(e,x) \wedge \text{has\_method}(x,y) \wedge \text{complete}(y,e)$$

When working with a number of related checklists it is important to determine which ones should be followed at any given time. In our example, steps `:t1` and `:t2` are ordered. This means that step `:t2`, "Prepare a pancake", should be performed after step `:t1`, "Receive a pancake order". Therefore, an employee should not start following the instructions on how to prepare a pancake before a pancake order has been placed. In other words, the check-boxes in environment `:e1` should not be checked until check-box `:l1` has been checked. The logic behind this observation is captured by the the concept of an *active* environment. The fact that an environment e is active is denoted by predicate $active(e)$ (Definition 4.7.17). If an environment is not active, then no action should be taken to complete the executions of that environment.

Definition 4.7.17. $active(e)$ Tasks in environment e can be considered for execution.

In the simplest case, an environment that does not have super environments can be considered active. This notion is captured by Axiom 14. Using this axiom, the environment `:e0` in our example can be inferred to be active.

Axiom 14. An environment without super-environments is active.

$$\text{active}(e) \Longleftarrow \neg\exists a.\text{sub\_env}(e,a)$$

More complex reasoning is required to determine that environment `:e1` is also active. In fact, we can observe that environment `:e1` is active because (1) its goal is the completion of task `:t2`, namely "Prepare a pancake", in environment `:e0` and (2) this task (`:t2`) is *ready* to be executed in environment `:e0`. An task x is ready in an environment e if and only if it can be executed in that environment. This information is captured by the predicate $ready(x,e)$ (Definition 4.7.18). We can observe that task `:t2` should be considered ready in environment `:e0` ($ready($`:t2`$,$`:e0`$)$) because task `:t1` is complete in that environment. If task `:t1` was not complete, then task `:t0` would not be ready.

Definition 4.7.18. $ready(x,e)$ Actions to accomplish task x can be performed in environment e. The logical definition of this predicate is given in Axiom 15.

Before providing a logical definition of the predicate *ready*(x,e), we need to provide a notion of dependency between tasks. In fact, we can say that task `:t2` depends on, or *requires*, step `:t1`. A task x can be said to require, or be dependent on, another task y if the non-completion of task y can violate the *dependency criteria* of task x. If the dependency criteria of a task are violated, then that task cannot be considered ready. This notion of dependency between tasks is not only used to define the ordering between steps, but it can also be used to represent other types of requirements, such as the ingredients of a recipe.

For example, the pancake recipe in our discussion requires a number of ingredients. From this we can infer that without the required ingredients the process of making a pancake is not ready to start. In PROHOW, the dependency criteria of a task depend on the predicates *requires*(y,x) (Definition 4.7.19), which defines cumulative dependency, and *requires_one*(y,x) (Definition 4.7.20), which defines sufficient dependency. The dependency criteria of a task x are met in an environment e only if one of the following conditions is met:

1. Task x does not depend on any other task.

$$\neg\exists y.\text{requires\_one}(x,y) \vee \text{requires}(x,y))$$

2. Task x has at least one cumulative dependency and all of the cumulative dependencies of x are complete in that environment.

$$\exists y.\text{requires}(x,y) \wedge \forall z.\text{requires}(x,z) \rightarrow \text{complete}(z,e)$$

3. One of the sufficient dependencies of x is complete in that environment.

$$\exists y.\text{requires\_one}(x,y) \wedge \text{complete}(y,e)$$

Definition 4.7.19. *requires*(x,y) Task x has a cumulative dependency on task x. This means that the completion of task y and all the other tasks z such that *requires*(x,z) is sufficient to meet the dependency criteria of x.

Definition 4.7.20. *requires_one*(x,y) Task x has a sufficient dependency on task y. This means that the completion of task y is sufficient to meet the dependency criteria of x.

In our example, we can notice that task task `:t0` does not have any requirements. Its dependency criteria can therefore be considered to be met. Task `:s0`, on the other hand, depends on a number of ingredients. We can say that task `:s0` has a cumulative dependency on tasks `:r1`, `:r2` and `:r3` by stating *requires*(`:s0`, `:r1`), *requires*(`:s0`, `:r2`) and *requires*(`:s0`, `:r3`). This captures the fact that no single one of these three ingredients is sufficient on its own, but that all three need to be available before the pancake preparation process can be ready to start. We can also say that task `:s0` has a sufficient dependency on task `:r4` by stating *requires_one*(`:s0`, `:r4`). This captures the fact that the ingredient "Pancake Mix" is sufficient, on its own, to start making pancakes. In other words, the process of making a pancake can be performed if all of the three ingredients: flour, eggs and milk are present, or if the pancake mix ingredient is available. In a similar fashion, the ordering between the steps `:t2` and `:t1` can be expressed by statement *requires*(`:t2`, `:t1`), meaning that step `:t1` must precede step `:t2`.

Based on this notion of dependency, Axiom 15 provides the logical definition of the notion of a ready task. More specifically, in order for a task x to be considered ready to be executed in an environment e, three conditions must hold: (1) its dependency criteria must be satisfied, (2) the environment e must be active and (3) x must not be a step of any other task, or it must be a step of task that is ready in the same environment. This last condition is necessary to make sure that the steps of a task are not executed before that task is ready. For example, the ingredients of the pancake recipe can be defined as requirements of the task `:s0`. Without the third condition step `:s1`, which does not have any direct dependencies, could be considered ready even before the ingredients are available.

Axiom 15. Logical definition of the predicate *ready*(x,e)

$$
\begin{aligned}
&\text{ready}(x,e) \iff \text{active}(e) \wedge \\
&((\neg\exists y.\text{has\_step}(y,x)) \vee (\exists y.\text{has\_step}(y,x) \wedge \text{ready}(y,e))) \wedge \\
&((\neg\exists y.\text{requires\_one}(x,y) \vee \text{requires}(x,y)) \vee \\
&(\exists z.\text{requires}(x,z) \wedge \forall y.\text{requires}(x,y) \rightarrow \text{complete}(y,e)) \vee \\
&(\exists y.\text{requires\_one}(x,y) \wedge \text{complete}(y,e)))
\end{aligned}
$$

Having defined the predicate *ready*(x,e), we can define when sub-environments can be considered active. Axiom 16 extends Axiom 14 into a sufficient and necessary definition of the predicate *active*(e). This axiom states that an environment is active if it does not have a super-environment, or if its goal task is ready in one of its active super-environments. In our example, the goal of environment `:e1`, namely `:t2`, is ready in its super-environment `:e0`, which is active. This fact allows us to conclude that environment `:e1` is also active.

Axiom 16. Logical definition of the predicate *active*(e)

$$\begin{aligned}\text{active}(e) \iff & (\neg\exists a.\text{sub\_env}(e,a)) \vee \\ & (\exists a.\text{sub\_env}(e,a) \wedge \text{has\_goal}(e,x) \wedge \text{ready}(x,a) \wedge \text{active}(a))\end{aligned}$$

The definition of ready also allows us to define how a task can be considered complete in an environment if it has been successfully completed in one of its sub-environments. As we have already seen, Axiom 13 can be used to infer that task `:t2` is complete in environment `:e1` once the pancake recipe represented by task `:s0` is complete in that environment. However, we now need a way to infer that task `:t2` should also be considered complete in the super-environment `:e0`. Axiom 17 states that if a task x is ready in environment e, and complete in an environment a that is a sub-environment of e and that has goal x, then x can also be considered complete in environment e. In our example, if we assume that task `:t2` is complete in environment `:e1`, then Axiom 17 allows us to infer that task `:t2` is also complete in environment `:e0`. This inference is possible because task `:t2` is the goal of environment `:e1` and because it is ready in environment `:e0`.

Axiom 17. Definition of how a task can be considered complete if it is complete in one of its sub-environments.

$$\text{complete}(x,e) \Longleftarrow \exists a.\text{ready}(x,e) \wedge \text{complete}(x,a) \wedge \text{sub\_env}(a,e) \wedge \text{has\_goal}(a,x)$$

In some instructions, it might be necessary to determine the real-world value of certain objects. For example, step `:t3` asks the employee to calculate the cost of the pancake that the client should pay. The cost could vary depending on the time of the day or on discount vouchers possessed by the client. In this situation it is not only useful to keep track of whether task `:t3` has been completed or not, but also to know the resulting value of this operation. It is possible to attach a resulting value v to an execution i using the *has_value*(i,v) predicate (Definition 4.7.21). The resulting value v of a task x completed in an environment e is denoted by predicate *value*(v,x,e) (Definition 4.7.22). Axiom 18 specifies that if a successful execution i of a task x is complete in an environment e and has value v, then v is the value of task x in that environment. In our example, the employee of the café might determine that the current price for a pancake is £3. He or she might then check off check-box `:i3` (*success*(`:i3`)) and annotate "£3" as the value of that execution (*has_value*(`:i3`, "£3")). The term "£3" is used here to refer to an entity representing the price of 3 pounds. Using Axiom 18, we can now infer that the value of the task `:t3` in environment `:e0` is 3 pounds (*value*("£3", `:t3`, `:e0`)).

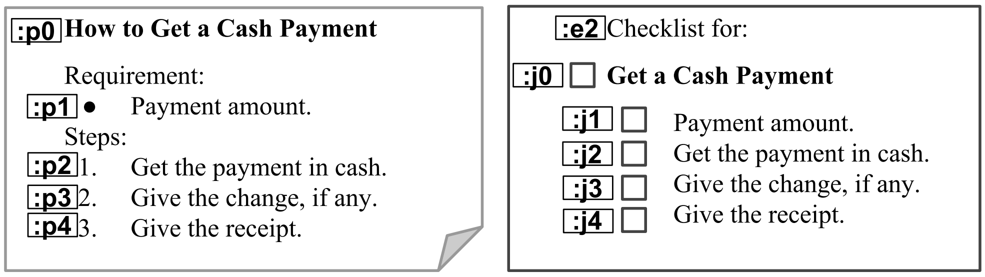

:p0 How to Get a Cash Payment
Requirement:
:p1 ● Payment amount.
Steps:
:p2 1. Get the payment in cash.
:p3 2. Give the change, if any.
:p4 3. Give the receipt.
:e2 Checklist for:
:j0 ☐ Get a Cash Payment
:j1 ☐ Payment amount.
:j2 ☐ Get the payment in cash.
:j3 ☐ Give the change, if any.
:j4 ☐ Give the receipt.

Figure 4.8: A possible visualisation of a set of instructions on how to accept cash payments, and associated checklist. The entities in this visualisation have been annotated with URI identifiers.

Definition 4.7.21. $has\_value(i, v)$ Execution i resulted in value v.

Definition 4.7.22. $value(v, x, e)$ Entity v is the resulting value of the completion of task x in environment e.

Axiom 18. The value of a task in an environment can be inferred by the value of an execution of that task in that environment.

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

To explain the remaining features of the PROHOW model, let us imagine that the manager of the café from our example decides to add an additional set of instructions on how to accept cash payments. A representation of this set of instructions and a corresponding checklist are presented in Figure 4.8. The set of instructions on how to get a cash payment are one method to complete the step “Get payment from the client” (has_method(`:t4`, `:p0`)). Environment `:e2` will now be considered a sub-environment of environment `:e0` (sub_env(`:e2`, `:e0`)) with goal `:t2` (has_goal(`:e2`, `:t4`)).

The set of instructions for task `:p0` require knowledge of the amount of money that the client should pay (task `:p1`). This amount is the same as the amount of money that an employee would have already calculated when completing task `:t3`. It is therefore useful to reuse this knowledge, to avoid calculating the same amount of money twice. In this case, it is said that task `:p1` *binds* with task `:t3`. This fact, captured by the predicate $binds(x, y)$, states that the execution of y should also count as the execution of x in related environments (Definition 4.7.23). The concept of bindings is needed because tasks which are complete in one environment cannot, by default, be considered complete in related environments. Axiom 19 defines how bindings can be used to infer the completion and value of tasks. If task x binds to task y ($binds(x, y)$) then task x can be considered complete in every environment e such that x is ready in e and y is complete in an environment a related to e. For example, we can imagine an employee of the café calculating the price for a pancake in environment `:e0` (*value*("£3", `:t3`, `:e0`)). By binding task `:p1` and task `:t3` (*binds*(`:p1`, `:t3`)) we can now use Axiom 19 to infer the value of task `:p1` in environment `:e2` (*value*("£3", `:p1`, `:e2`)).

Definition 4.7.23. *binds*(x,y) Task x can be considered complete in an environment if task y is complete in a related environment. The value of y, if present, is also the value of x.

Axiom 19. Definition of how bindings can be used to infer the completion and the value of a task.

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

If known a priori, certain values can be predefined using *constants*. Predicate *has_constant*(x,y) states that task x is associated with a task y which has also y as the value (Definition 4.7.24). In this case, y acts both as a task and, more generally, as a value. For example, the manager of the café might decide that pancakes should always cost the same amount, for example 4 pounds. This constant can be attached to the task `:t0` of serving a pancake using statement *has_constant*(`:t0`, "£3"). Bindings can then be used to specify that this constant value should be used in place of task `:t3`, namely "Calculate the cost" (*binds*(`:t3`, "£3")). Once these two statements are asserted, task `:t3` can be inferred to be complete as soon as it becomes ready, and its value can be inferred to be "£3". If tasks `:p1` and `:t3` are bound (*binds*(`:p1`, `:t3`)), this value can also be inferred as the value that corresponds to the payment amount (task `:p1`) of the set of instructions on how to make a client pay by cash (task `:p0`).

Definition 4.7.24. *has_constant*(x,y) Whenever task x is the task of an execution in an environment, task y can also be considered accomplished in that environment and as having value y.

Axiom 20. Definition of how the completion of a task and its value can be inferred using constants.

$$\text{complete}(x,e) \wedge \text{value}(x,x,e) \Longleftarrow \exists i,y.\text{has\_env}(i,e)$$
$$\wedge\ \text{has\_task}(i,y) \wedge \text{has\_constant}(y,x)$$

The previous Axioms are combined into sufficient and necessary definitions for the *complete* and *value* predicates, the only two derived predicates that have not been completely defined yet. Axiom 21 provides a sufficient and necessary definition for the predicate *complete* by combining Axioms 10, 12, 13, 17, 19 and 20.

Axiom 21. Sufficient and necessary definition of the *complete* predicate.

$$\begin{aligned}
&\text{complete}(y,e) \iff (\exists i.\text{has\_env}(i,e) \land \text{has\_task}(i,y) \land \text{success}(i)) \lor \\
&((\exists x.\text{has\_step}(y,x)) \land (\forall x.\text{has\_step}(y,x) \rightarrow \text{complete}(x,e))) \lor \\
&(\exists x.\text{has\_goal}(e,y) \land \text{has\_method}(y,x) \land \text{complete}(x,e)) \lor \\
&(\exists a.\text{ready}(y,e) \land \text{complete}(y,a) \land \text{sub\_env}(a,e) \land \text{has\_goal}(a,y)) \lor \\
&(\exists x,a.\text{ready}(y,e) \land \text{binds}(y,x) \land \text{complete}(x,a) \land \text{related}(a,e)) \lor \\
&(\exists i,x.\text{has\_env}(i,e) \land \text{has\_task}(i,x) \land \text{has\_constant}(x,y))
\end{aligned}$$

Axiom 22 provides a sufficient and necessary condition for the predicate *value* by combining Axioms 18, 19 and 20.

Axiom 22. Sufficient and necessary definition of the *value* predicate.

$$\begin{aligned}
&\text{value}(z,y,e) \iff \\
&(\exists i.\text{has\_env}(i,e) \land \text{has\_task}(i,y) \land \text{success}(i) \land \text{has\_value}(i,z)) \lor \\
&(\exists x,a.\text{ready}(y,e) \land \text{binds}(y,x) \land \text{value}(z,x,a) \land \text{related}(a,e)) \lor \\
&(\exists i,x.\text{has\_env}(i,e) \land \text{has\_task}(i,x) \land \text{has\_constant}(x,y) \land z = y)
\end{aligned}$$

The predicates of this model can be used to infer the types of their variables as illustrated in Figure 4.2. These typing inferences are formalised by Axiom 23.

Axiom 23. Definition of the types associated with the predicates of the model.

$$\begin{aligned}
&\text{has\_step}(x,y) \rightarrow \text{task}(x) \land \text{task}(y) \\
&\text{has\_method}(x,y) \rightarrow \text{task}(x) \land \text{task}(y) \\
&\text{requires}(x,y) \rightarrow \text{task}(x) \land \text{task}(y) \\
&\text{requires\_one}(x,y) \rightarrow \text{task}(x) \land \text{task}(y) \\
&\text{has\_task}(i,x) \rightarrow \text{execution}(i) \land \text{task}(x) \\
&\text{has\_env}(i,e) \rightarrow \text{execution}(i) \land \text{environment}(e) \\
&\text{has\_goal}(e,x) \rightarrow \text{environment}(e) \land \text{task}(x) \\
&\text{success}(i) \rightarrow \text{execution}(i) \\
&\text{sub\_env}(e,a) \rightarrow \text{environment}(e) \land \text{environment}(a)
\end{aligned}$$

$$\text{has\_value}(e,v) \rightarrow \text{execution}(x)$$

$$\text{has\_constant}(x,y) \rightarrow \text{task}(x) \wedge \text{task}(y)$$

$$\text{binds}(x,y) \rightarrow \text{task}(x) \wedge \text{task}(y)$$

$$\text{finished}(e) \rightarrow \text{environment}(e)$$

$$\text{active}(e) \rightarrow \text{environment}(e)$$

$$\text{complete}(x,e) \rightarrow \text{task}(x) \wedge \text{environment}(e)$$

$$\text{ready}(x,e) \rightarrow \text{task}(x) \wedge \text{environment}(e)$$

$$\text{related}(e,a) \rightarrow \text{environment}(e) \wedge \text{environment}(a)$$

$$\text{value}(v,x,e) \rightarrow \text{task}(x) \wedge \text{environment}(e)$$

Conceptually, the predicates of the PROHOW model could be divided in two groups, namely *static* and *dynamic* PROHOW predicates. The former are used to represent sets of instructions, while the latter are used to represent their executions. Static predicates, which only describe Task entities, are: task(x), has_step(x,y), has_method(x,y), requires(x,y), requires_one(x,y), has_constant(x,y) and binds(x,y). These predicates alone are sufficient to represent sets of instructions. These predicates are called "static" because a knowledge base containing representations of instructions does not need to be updated frequently. However, changes to these static predicates are possible. For example, the creation of a new set of instructions, or the modification of an existing set of instructions would result in the addition or deletion of facts based on those static predicates.

All the predicates of the PROHOW model that are not static are dynamic. Dynamic predicates are used to represent the execution of a set of instructions by describing Execution and Environment entities. These predicates are called "dynamic" because a representation of the execution of a set of instructions needs to be constantly updated. In fact, whenever this execution progresses, new facts described using these dynamic predicates should be asserted.

The FOL axioms defined in this section lay the semantic foundation of the PROHOW knowledge representation for the rest of this thesis. The main use of these axioms is to provide a formal definition to the terms of the PROHOW model. In many lightweight vocabularies, such as Schema.org, terms are mostly defined by natural language descriptions. Such definitions are easy for humans to interpret, but they are prone to misinterpretation, and they do not allow machines to reason about them automatically. The PROHOW model allows automated reasoning and its FOL axioms define the inference rules that machines can perform. When a set of instructions or a checklist described using the PROHOW model is processed by a machine, these rules can be used to infer new facts.

In particular, the logical definitions of the PROHOW model will be used to answer the competency questions introduced in Section 4.4. The computational solution to answer these competency questions, which will be presented in Section 4.10, is based on the FOL definitions of the PROHOW terms. For example, Axiom 15 will be used to determine whether a task is *ready*. This inference will be used to determine which tasks are ready to be executed in an environment to answer competency question `Q5`.

This computational solution can be used in concrete applications of the PROHOW model. For example, it is used in the human-machine collaboration application that will be presented in Section 7.3. This application uses the FOL definitions of the PROHOW model to reason about concrete to-do checklists and infer, for example, that a task has been completed if all of its steps are complete.

4.7.3 Rewriting of the Competency Questions

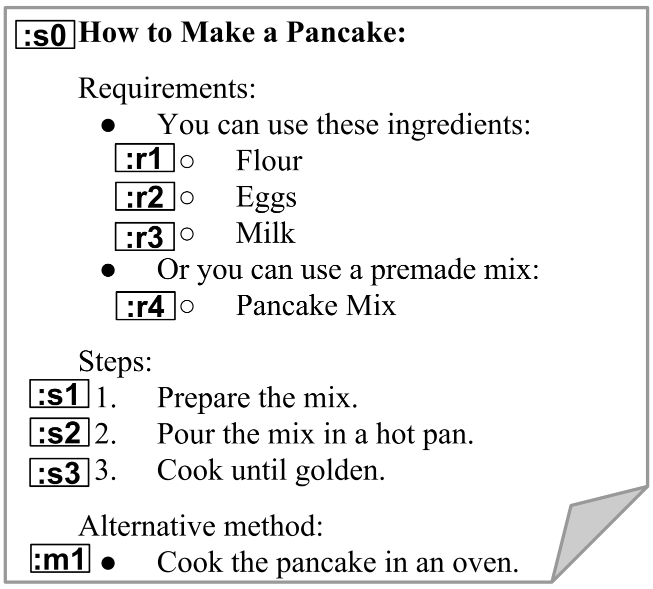
The competency questions that guide the development of this knowledge representation model have been previously described in natural language in Section 4.4. I will now redefine these competency questions with respect to the terminology of the PROHOW model. The competency questions related to the scenario of a user exploring a set of instructions is the following:

Q1 What are the requirements of a task x? Find the set R^x_{all} of all the sets R^x such that the completion of every task in R^x allows x to be ready for execution, while no proper subset of R^x does. If a task does not have requirements, than R^x_{all} is empty. The answer to this question, namely the set R^x_{all}, is a set of sets of tasks.

Q2 How can task x be decomposed into steps? Find the set S^x_{all} of all the sets S^x such that the complete execution of every task in S^x is equivalent to the completion of x, while no proper subset of S^x does. The answer to this question, namely the set S^x_{all}, is a set of sets of tasks.

To explain these first two competency questions, we can imagine the set of instructions in Figure 4.9. This figure extends the set of instructions presented in Figure 4.6 with an additional method `:m1`, namely “Cook the pancake in an oven”, which can be used to complete the main task `:s0`.

Given this set of instructions, competency question `Q1` with respect to task `:s0` is answered by set $R_{all}^{\texttt{:s0}} = \{\{\texttt{:r1}, \texttt{:r2}, \texttt{:r3}\}, \{\texttt{:r4}\}\}$. In other words, the requirements of `:s0` are either the set of ingredients `:r1`: "Flour", `:r2`: "Eggs" and `:r3`: "Milk", or the ingredient `:r4`: "Pancake Mix" instead. Competency question `Q2` with respect to task `:s0`, is answered by set $S_{all}^{\texttt{:s0}} = \{\{\texttt{:s1}, \texttt{:s2}, \texttt{:s3}\}, \{\texttt{:m1}\}\}$. The first and second set of tasks in S_{all}^{t} represent, respectively, the list of steps, and the alternative method of `:s0`. I will assume these steps need to be performed in the given order. For example, task `:s2` requires the completion of task `:s1` ($R_{all}^{\texttt{:s2}} = \{\{\texttt{:s1}\}\}$.


$\boxed{:s0}$ **How to Make a Pancake:**

Requirements:

- You can use these ingredients:
  - $\boxed{:r1}$ Flour
  - $\boxed{:r2}$ Eggs
  - $\boxed{:r3}$ Milk
- Or you can use a premade mix:
  - $\boxed{:r4}$ Pancake Mix

Steps:

$\boxed{:s1}$ 1. Prepare the mix.
$\boxed{:s2}$ 2. Pour the mix in a hot pan.
$\boxed{:s3}$ 3. Cook until golden.

Alternative method:

$\boxed{:m1}$ ● Cook the pancake in an oven.

Figure 4.9: Extended version of the set of instructions presented in Figure 4.6.

The competency questions related to the scenario of a user accessing a to-do checklist based on a set of instructions is the following:

Q3 Which tasks are in the checklist? Find the set T^e of all the tasks that are scheduled to be executed in a checklist e. The answer to this question, namely the set T^e, is a set of tasks.

Q4 Which tasks have been completed in the checklist? Find the set T_{compl}^{e} of all the tasks that are completed in a checklist e. The answer to this question, namely the set T_{compl}^{e}, is a set of tasks.

Q5 Which tasks are ready to be completed in the checklist? This question can be derived from questions `Q1`, `Q3` and `Q4`. Find the set T_{ready}^{e} of all the tasks x in T^e such that (1) they do not have requirements: $R_{all}^{x} = \varnothing$ or (2) there is at least one set R^x in its requirement sets R^x_{all} (computed using question `Q1`) such that R^x is included in the set T^e_{compl} of all the completed tasks (computed using question `Q4`) in a given environment e. The answer to this question, namely the set T^e_{ready}, is a set of tasks.

Q6 Which tasks are being completed by following a separate checklist? Find the set E^e of pairs $< x, a >$ such that the complete execution of checklist a can be considered as equivalent to the completion of task x in checklist e. The answer to this question, namely the set E^e, is a set of pairs of tasks and environments.

Q7 Which tasks can be completed at the current time to advance the progress toward the completion of the checklist? This question can be derived from questions `Q4`, `Q5`, `Q6`. Find the set T^e_{next} of incomplete tasks which are ready to be executed: $T^e_{next} = T^e_{ready} \backslash T^e_{compl}$ within an environment e. This question can also be recursively extended to all the sub-environments a found in the pairs $< x, a >$ of set E^e from question `Q6` if $x \in T^e_{next}$. The answer to this question, namely the set T^e_{next}, is a set of tasks.

Q8 How can a checklist e to complete task t be generated? Find a partially ordered set of tasks C such that (1) the execution of these tasks in the given order does not violate the dependencies of these tasks, (2) the completion of all these tasks in environment e is equivalent to the completion of t in e and (3) no proper subset of $C \backslash \{t\}$ has properties (1) and (2). Possible answers to this question can be represented as lists of sets of tasks, such that the tasks in a set can be completed in any order, but only after all the tasks in all of the antecedent sets in the list have been completed.

An example of a checklist `:e3` generated from the set of instructions on how to accomplish `:s0` is shown in Figure 4.10. This checklist corresponds to the list of sets: [{`:r1`, `:r2`, `:r3`}, {`:s1`}, {`:s2`}, {`:s3`}]. This list dictates that tasks `:r1`, `:r2` and `:r3` can be completed in any order before completing task `:s1`. After `:s1`, task `:s2` can be completed, and after that `:s3`. When all of these tasks are complete, the main task `:s0` can also be considered complete. This list is a correct answer to question `Q8` as it represents a partially ordered set of tasks that fulfils the criteria defined by this question. Other answers are also possible, such as the list: [{`:r4`}, {`:m1`}], which corresponds to the actions of cooking a pancake in an oven using a pre-made pancake mix.

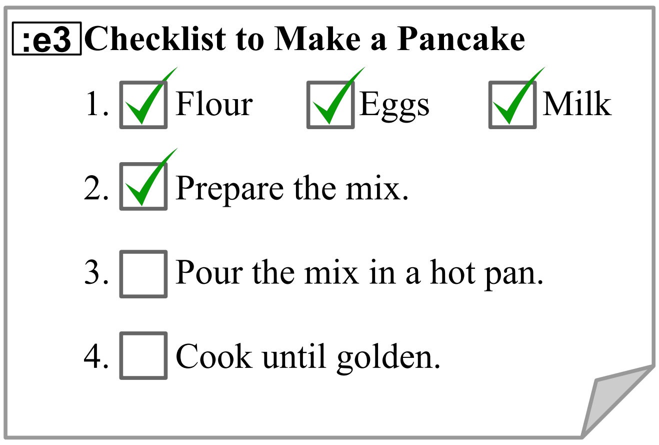


Figure 4.10: Example of a partially completed checklist derived from the set of instructions in Figure 4.9.

With respect to the checklist in Figure 4.10, the answer to competency question Q3 is the set of tasks $T^{:e3} = \{:r1, :r2, :r3, :s1, :s2, :s3\}$. The answer to question Q4 is the set of complete tasks $T^{:e3}_{compl} = \{:r1, :r2, :r3, :s1\}$. The answer to question Q5 is $T^{:e3}_{ready} = \{:r1, :r2, :r3, :s1, :s2\}$, as :s3 is the only task in the checklist that does not have all of its requirements satisfied. If a separate checklist :e4, describing how to complete task :s3 in more details, was being followed, then the answer to question Q6 would be: $E^{:e3} = \{< :s3, :e4 >\}$. The set of tasks that can be accomplished next to progress the execution of the checklist :e3, which provides the answer to question Q7, is the set $\{:s2\}$.

Competency question Q8 can be seen close to the notion of automated planning, and in particular to HTN planning. Similarly to planning, question Q8 involves the search for a sequence of simpler tasks to decompose a more complex task. The goal of accomplishing the complex task can be seen as the goal of an Automated Planning (AP) problem. Moreover, question Q8 requires the ordering of tasks in a way that does not violate certain constraints. In AP, a goal might not be satisfiable, for example if there is a deadlock in the requirements of a task, or multiple solutions might exist. A user capable of performing any task can complete a checklist simply by completing the tasks of the checklist one by one in the given order.

Given that questions Q5 and Q7 can be derived by the other questions, the set of competency questions required by a knowledge representation formalism to model the domain of human instructions is made of questions Q1, Q2, Q3, Q5, Q6 and Q8.

4.7.4 Workflow Interpretation of the Model

The expressiveness of the PROHOW model goes beyond simple step sequences which are common in step-by-step instructions. In fact, this model can express all the basic control flow patterns defined by Van Der Aalst et al. (2003).

Sequence Pattern This pattern is defined as follows: "An activity in a workflow process is enabled after the completion of another activity in the same process" (Van Der Aalst et al. 2003). An example of this pattern is provided in Figure 4.11. The fact that an activity x is enabled after the completion of another activity y can be expressed by the following PROHOW statement:

$$\text{requires}(x, y)$$

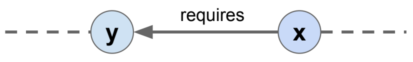


Figure 4.11: Sequence workflow pattern example: task x is executed after task y.

Synchronization This pattern is defined as follows: "A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple thread." (Van Der Aalst et al. 2003). An example of this pattern is provided in Figure 4.12. The fact that multiple threads of execution of which the set of their last activities in each thread is Y converge to a point in the workflow that immediately precedes a set of activities X can be expressed by the following PROHOW statement:

$$\forall y \in Y, x \in X.\text{requires}(x, y) \tag{4.1}$$

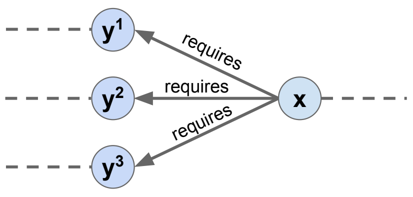


Figure 4.12: Synchronization workflow pattern example: task x can be executed after the completion of all the tasks y^1, y^2 and y^3, which can be executed in parallel.

Simple Merge This pattern is defined as follows: "A point in the work ow process where two or more alternative branches come together without synchronization" (Van

Der Aalst et al. 2003). An example of this pattern is provided in Figure 4.13. The fact that a point in the workflow that immediately precedes a set of activities X represents a non-synchronized merge of multiple threads of execution of which the set of their last activities in each thread is Y can be expressed by the following PROHOW statement:

$$\forall y \in Y, x \in X.\text{requires\_one}(x, y) \tag{4.2}$$

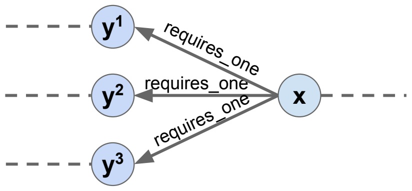


Figure 4.13: Simple Merge workflow pattern example: task x can be executed after the completion of any of the tasks y^1, y^2 and y^3.

Parallel Split This pattern is defined as follows: “A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order” (Van Der Aalst et al. 2003). A partial example of this pattern is provided in Figure 4.14. The fact that a point in the workflow that immediately follows a set of activities X splits into multiple threads of which the set of their initial activities in each thread is Y can be expressed by the following PROHOW statement *only if* the threads later converge using the Synchronization pattern.

$$\forall y \in Y, x \in X.\text{requires}(y, x) \tag{4.3}$$

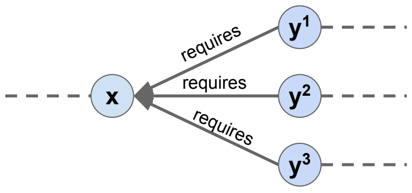


Figure 4.14: Partial example of the Parallel Split and Exclusive Choice workflow patterns: tasks y^1, y^2 and y^3 can be executed after task x.

Exclusive Choice This pattern is defined as follows: “A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen” (Van Der Aalst et al. 2003). This pattern is implemented in the same way as the Parallel Split pattern. In the workflow interpretation of PROHOW, in fact, whether all of the branches need to be followed or not, is implicit in the way they converge. If they converge using a Synchronization pattern, as depicted in Figure 4.16, then it can be inferred that all of the branches need to be followed. If they converge using a Simple Merge pattern instead, as depicted in Figure 4.15, then it can be inferred that only one of the branches needs to be followed.

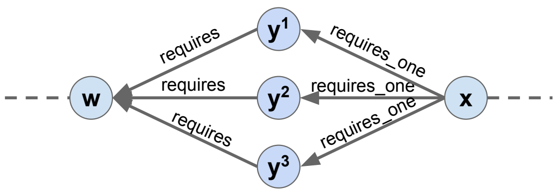


Figure 4.15: Exclusive Choice workflow pattern example: by using the Simple Merge pattern, it is implicit that only one of tasks y^1, y^2 and y^3 needs to be complted after task w and before task x.

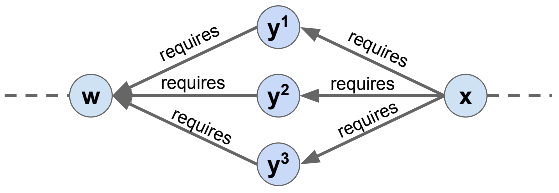


Figure 4.16: Parallel Split workflow pattern example: by using the Synchronisation pattern, it is implicit that all of the tasks y^1, y^2 and y^3 need to be completed after task w and before task x.

4.7.5 Logical Properties of the Model

This logical model describes the state of the world in terms of a known set of facts W, consisting only of the base predicates of the model, which state which facts are true in the state of the world. Facts which are not true in the state of the world are not expressed directly.

Two important assumptions related to the availability of facts can be made: the Open-World Assumption (OWA) and the Closed-World Assumption (CWA) (Minker 1982). The CWA implies complete knowledge of the world. In other words, if a statement cannot be inferred true from the set of statements in W, then it can be assumed to be false. Under the OWA, instead, the state of the world is considered to be only partially observable. Therefore, if a statement x cannot be inferred from W, it is not considered false, but *unknown* instead, as it is possible that certain true but unknown statements can prove x. Since perfect knowledge of the domain is hard to achieve, many real world situations can be described as following the OWA.

The main problem of the OWA is the difficulty in proving that a certain statement is false in the state of the world. An axiom without negation whose terms are bound by the existential quantifier can be proven true in the OWA if it is expressed in W or if it can be derived from it. On the other hand, the same axiom cannot be proven true if its terms are bound by the universal quantifier. This is a result of the fact that an axiom bound by an universal quantifier is only true if there is no counter-example, namely a possible assignment of variables which does not satisfy the axiom. Under the OWA, unless the axiom under the universal quantifier is a tautology or a contradiction, it is always possible that a counter-example exists. For example, part of Axiom 15 verifies whether all of the requirements of a task are complete. Under the OWA it is impossible to prove this part of Axiom 15, as it is always possible that there is an unknown dependency to a task which is incomplete. In other words, under the OWA it is not possible to prove that all of the requirements of a task have been completed or that all of its steps have been completed.

While the OWA does not allow important inferences of the model to be made, the CWA is impractical to assume in a real context. For this reason, I formulate a third assumption that I call Semi-Open-World Assumption (SOWA). The SOWA is an OWA with one added constraint. Under the SOWA, complete knowledge of the state of the world $\bar{W}$ is divided into a number of subsets, or resources, R, such that $\bigcup_{r \in R} r = \bar{W}$, and each task t is associated with a set of resources $D(t)$ which contain exhaustive information about task t. More specifically, each step $has\_step(t,x)$ and each requirement $requires(t,x)$ of task t needs to be explicitly asserted in each resource in $D(t)$. The axioms in 4.4 formalise this property using predicate $contains(a,b)$ to state that statement b is explicitly asserted in resource a.

$$\begin{aligned} &\forall t, d \in D(t).\text{has\_step}(t,x) \longrightarrow \text{contains}(d, \text{has\_step}(t,x)) \\ &\forall t, d \in D(t).\text{requires}(t,x) \longrightarrow \text{contains}(d, \text{requires}(t,x)) \end{aligned} \tag{4.4}$$

Under the SOWA, the inferences described in Axiom 12 and 15 can be made for a task t as long as all the facts contained in one of the resources $D(t)$ are known. If at least one of the resources $D(t)$ is included in the database, SOWA differs from OWA in the sense that failure to prove $has\_step(t,x)$ or $requires(t,x)$ proves, respectively $\neg has\_step(t,x)$ and $\neg requires(t,x)$.

While not as general as OWA, SOWA captures the common sense intuition that exhaustive information about the steps and the requirements of a particular task should be contained in a single resource. For example, a typical web user interested in preparing a pancake might open a web-page containing a recipe for making pancakes. Arguably, such a user would follow the instructions assuming that all the requirements and steps of the recipe are exhaustively listed on that page. This assumption is equivalent to SOWA, and it allows the user to reliably make universally quantified inferences over the task described in the web-page, without having to assume complete knowledge of the world (CWA).

It is important to state that the concept of *exhaustive* is subjective to the information needs of the recipient of such information. For example, a pancake recipe might not specify that “a source of heat” is required to cook the pancake. Since this information can be thought of as implicit from the point of view of human common-sense, its omission from a set of instructions does not contradict the SOWA assumption.

4.7.5.1 Execution-Equivalent Instructions

If we are interested in achieving a given task, we might find multiple sets of instructions online that explain how to accomplish it. Some of them might describe different approaches, or more specifically a different sequence of actions to accomplish the task. In this case I define the sets of instructions as being *semantically* different. Other sets of instructions might describe the same sequence of actions, but representing it in different ways. In this last case, I define these sets of instructions as being *semantically* equivalent but *structurally* different.

These notions of equivalence are applicable to sets of instructions which contain the same minimal set of execution tasks M^t. M^t is the set of tasks that are included in all the the possible *checklist traces* E^t to accomplish task t. Each checklist trace in E^t is an ordered list of tasks, such that accomplishing these tasks in the given order results in accomplishing the main task t. Given T^e as the set of tasks in the checklist trace e, M^t is defined by Formula 4.5.

$$M^t = \bigcup_{e \in E^t} T^e \tag{4.5}$$

. If two sets of instructions contain a different minimal set of execution tasks, then they can be said to be semantically different, as they require the completion of different actions. It is important to notice that this definition of equivalence assumes that if two tasks are the same, then they are represented by the same variable. Determining whether two tasks are the same is a separate problem, not considered in these definitions.

I define two sets of instructions to accomplish a task t to be *semantically equivalent* if and only if they result in the same set of checklists traces E^t. Two sets of instructions are said to be *structurally equivalent* if and only if they are defined by the same set of axioms.

To illustrate these notions of equivalence, we can consider the three sets of instructions in Figure 4.17, which represent a simplified version of the set of instructions presented earlier in the chapter. In this example, task `:t0` has only 2 steps. The minimal set of execution tasks M^x and the set of all possible checklists E^x is also given in the figure. All three sets of instructions contain the same minimal set of executions task: $\{$`:t0`, `:t1`, `:t2`$\}$. These sets of instructions are not structurally equivalent, as they are defined by a different set of axioms. For example, the axiom `requires`(`:t2`, `:t1`) is only included in Version 1, while the axiom `requires_one`(`:t2`, `:t1`) is only included in Version 2. Versions 1 and 2 are semantically equivalent, as they can be interpreted as the same set of possible checklists E^x. Version 3, however, is not semantically equivalent to the other versions. Lacking a dependency between tasks `:t2` and `:t1`, Version 3 allows the completion of task `:t2` before task `:t1`. This results in the checklist trace [`:t2`, `:t1`], which cannot be computed from the other two versions.

The notion of semantic equivalence can be used to determine whether two sets of instructions are, in practice, interchangeable. Two semantically equivalent but structurally different sets of instructions might behave differently when new knowledge is added. For example, adding the same axiom `requires`(`:t2`, `:t3`) to versions 1 and 2 would make them semantically different. In Version 1, task `:t2` would then require both tasks `:t1` and `:t3`. In Version 2, however, only one of these two tasks would be required.

4.8 Fifth Phase: a Concrete Vocabulary for Instructions

In the fifth and last phase of this method, I present a concrete vocabulary to represent the PROHOW model defined in the previous phase. To do so, I reuse existing data representation standards from the Semantic Web field that I reviewed in Section 2.1.

The PROHOW model has been expressed in First Order Logic (FOL) but it cannot be completely translated into Description Logics (DL), a logic formalism commonly used to describe ontologies (Baader et al. 2008), such as OWL (Bechhofer et al. 2004).

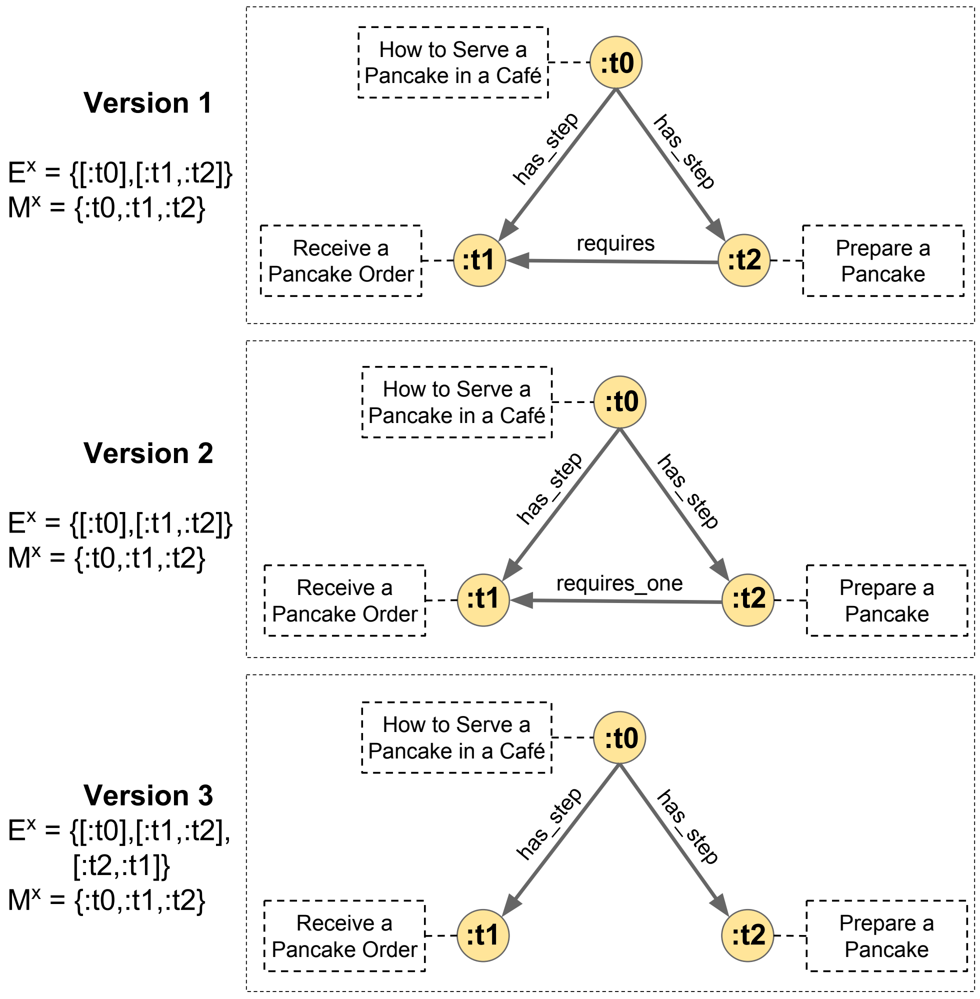


Figure 4.17: Three different sets of instructions for the same task. Only Versions 1 and 2 are semantically equivalent, while none of them is structurally equivalent.

Some aspects of the PROHOW model can be described in DL. More specifically, it is possible to model in OWL the list of classes and relations used in the PROHOW model, along with information about the domain and range of the relations. As we will see later, this will be done by adopting the RDFS vocabulary, which is used by the OWL language.

Most of the PROHOW model, however, is expressed as logical axioms, such as those that provide a necessary and sufficient definition for certain terms (e.g. Axiom 15). Many of these axioms cannot be translated into DL. DL was created as a less expressive formalism than FOL in order to allow tractable reasoning, and therefore only specific kinds of FOL expressions can be converted into DL. It has been shown that the *two variable fragment* and the *guarded fragment* of FOL can be translated into DL (Baader et al. 2008). However, the PROHOW model uses FOL axioms, such as

Axiom 10, which are not compatible with the two variable fragment of FOL because they contain more than two variables. Moreover, Axiom 10 does not belong to the guarded fragment because the predicates within the scope of the existential quantifier do not include all the free variables.

To create a lightweight instantiation of the PROHOW model, I choose to de-couple it from its axioms and to represent it as an RDFS vocabulary, which I call the PROHOW *vocabulary*. This RDFS vocabulary is a collection of URI terms that denote classes and relations that can represent all the logical statements of the model defined in the previous phase. The terms of this vocabulary are listed in Table 4.6, and follow the namespaces described in Table 2.1. While different naming conventions exist, this vocabulary follows the convention of only using lower case letters and of using underscores as word delimiters (Montiel-Ponsoda et al. 2011). These terms can be used to model all the 15 basic predicates of the PROHOW model listed in Table 4.4, and therefore also to infer the remaining 6 derived predicates.

Concept	Definition if α has this as its `rdf:type`
`prohow:task`	task(α)
`prohow:execution`	execution(α)
`prohow:environment`	environment(α)
Relation	**Definition if subject α has this relation to object β**
`prohow:requires`	requires(α, β)
`prohow:requires_one`	requires_one(α, β)
`prohow:has_step`	has_step(α, β)
`prohow:has_method`	has_method(α, β)
`prohow:has_task`	has_task(α, β)
`prohow:has_goal`	has_goal(α, β)
`prohow:binds_to`	binds(α, β)
`prohow:has_constant`	has_constant(α, β)
`prohow:has_value`	has_value(α, β)
`prohow:has_result`	success(α) if β is `prohow:complete`
`prohow:has_environment`	has_env(α, β)
`prohow:sub_environment_of`	sub_env(α, β)

Table 4.6: The concepts and relations of the PROHOW vocabulary.

A full specification of the PROHOW vocabulary is available in Appendix A as an OWL file. This vocabulary has been validated with the OntOlogy Pitfall Scanner! (OOPS!), developed by Poveda-Villalón et al. (2014). The OOPS! tool automatically validates an ontology against a number of common problems, or *pitfalls*, that were identified by analysing a corpus of 693 ontologies. The OOPS! tool did not detect any structural problem in the PROHOW vocabulary. However, it detected a failure in retrieving the vocabulary file through content negotiation. This problem might be due to a lack of support in URL redirection by the OOPS! tool, as the PROHOW vocabulary can be correctly retrieved in both human and machine understandable format using content negotiation. The *curl*[14] commands given in listings 4.1 and 4.2 demonstrate how it is possible to use the vocabulary URI to retrieve, respectively, a human and a machine understandable representation.

Listing 4.1: *curl* command to retrieve a human readable representation of the PROHOW vocabulary.

```
curl -L -H "Accept: text/html" http://w3id.org/prohow#
```

Listing 4.2: *curl* command to retrieve a machine readable representation of the PROHOW vocabulary.

```
curl -L -H "Accept: application/rdf+xml" http://w3id.org/prohow#
```

The number of triples necessary to represent a set of instructions using the PROHOW vocabulary has a low complexity, as this number scales linearly with the number of steps (α), requirements (β) and methods (γ) that the set of instructions contains. I will now provide an upper bound on the number of triples necessary to represent a set of instructions t, assuming that all the entities are labelled, and assuming that each step must be preceded by at most another step. This last condition is true for all linear sequences of steps, such as those found in the wikiHow and Snapguide repositories, because each step has at most one antecedent.

To represent a set of instructions t, one triple is required to define the label of t. Each requirement is defined by two triples, (1) the first to link the URI of the requirement to its label, and (2) the second one to link it to the task that requires it. Each step is defined by up to three triples. (1) the first to link the URI of the step to its label, (2) the second to link the step to the higher-level task that the step is decomposing and (3) the third, optional triple to link the step to another step that must precede it in the order of execution. Each method is defined by two triples: (1) the first to link the URI of the method to its label and (2) the second to link it to the higher-level task of which it is a method. Therefore, the number of triples to represent the set of instructions t is at most

[14]`https://curl.haxx.se/` (accessed on 17/10/2017)

$2\beta 3\alpha 2\gamma + 1$, which corresponds to complexity $O(\beta\alpha\gamma)$. Assuming that any representation of t needs to have at least $\beta\alpha\gamma$ triples, and therefore complexity $O(\beta\alpha\gamma)$, to be able to define a label for each entity, the complexity of the PROHOW representation can be said to be minimal.

Having already described the meaning of each term in the previous section, what will follow is a series of examples that demonstrate how RDF data following the PROHOW vocabulary materialises in practice. The following text snippet describes the pancake recipe of Figure 4.9, identified with the URI `:s0` as a PROHOW task. Throughout this thesis, RDF data will be presented in Turtle format using the RDF namespaces defined in Table 2.1. Following the most common standards for publishing data on the web, hierarchical relations, such as class membership, will be described using the RDFS vocabulary, while the natural language description of entities will be provided using the `rdfs:label` relation.

Listing 4.3: A pancake recipe defined as a PROHOW task.

```
:s0 rdf:type prohow:task .
:s0 rdfs:label "How to Make a Pancake".
```

The following example describes three steps of a pancake recipe, namely: (1) preparing the pancake mix, (2) pouring the mix into a hot pan and (3) cooking until the pancake is golden. Although all of these steps also belong to the PROHOW class of tasks, this class membership can be omitted as all the core PROHOW relations `prohow:has_step`, `prohow:has_method` and `prohow:requires` have RDFS domain and range instances of class `prohow:task`. In other words, whenever one of these relations is found in an RDF triple, both the subject and the object of such triple can be assumed to belong to class `prohow:task`.

Listing 4.4: The PROHOW representation of the steps of a pancake recipe.

```
:s0 prohow:has_step :s1, :s2, :s3 .
:s1 rdfs:label "Prepare the mix" .
:s2 rdfs:label "Pour the mix in a hot pan" .
:s3 rdfs:label "Cook until golden" .
```

In this example, the three steps of the pancake recipe are ordered. This is achieved by requiring step `:s3` to execute after step `:s2`, which in turn should execute after step `:s1`.

Listing 4.5: The PROHOW representation of the ordering of three steps.

```
:s3 prohow:requires :s2 .
:s2 prohow:requires :s1 .
```

In this other example, the dependency relation is used to define three ingredients of the recipe: (1) eggs, (2) milk and (3) flour, as its requirements.

Listing 4.6: The PROHOW representation of the requirements of the pancake recipe.

```
:s0 prohow:requires :r1, r2, r3 .
:r1 rdfs:label "Flour" .
:r2 rdfs:label "Eggs" .
:r3 rdfs:label "Milk" .
```

The following example defines the alternative method `:m1` to cook a pancake.

Listing 4.7: The PROHOW representation of the alternative method of the pancake recipe.

```
:s0 prohow:has_method :m1 .
:m1 rdf:label "Cook the pancake in an oven" .
```

The following example defines a new checklist `:e5` to prepare a pancake.

Listing 4.8: The PROHOW representation of a checklist to complete task `:s0`

```
:e5 rdf:type prohow:environment .
:e5 prohow:has_goal :s0 .
```

Given the previous PROHOW descriptions of this task, it can be inferred that the task `:s0` is not ready to be executed in environment `:e5`, since it depends on tasks which have not been accomplished. If they are accomplished, the following statements can be added to the database:

Listing 4.9: The PROHOW representation of the requirements of the pancake recipe accomplished in the environment.

```
:w1 prohow:has_environment :e5 .
:w1 prohow:has_task :r1 .
:w1 prohow:has_result prohow:complete .
:w2 prohow:has_environment :e5 .
:w2 prohow:has_task :r2 .
:w2 prohow:has_result prohow:complete .
:w3 prohow:has_environment :e5 .
:w3 prohow:has_task :r3 .
```

```
:w3 prohow:has_result prohow:complete .
```

After these statements have been added, it will be possible to infer that the task `:s0` is ready to be executed in environment `:e5`.

4.9 Compatibility with Existing Languages Evaluation: a Conversion to PDDL

The PROHOW model is inspired by previous work on formal process representations and its concepts can be translated into other process formalisms. To demonstrate this point, the PROHOW model has been translated into a PDDL domain definition. This domain definition is listed in its entirety in Appendix B. The PDDL version of the PROHOW model has been generated by translating the axioms of the model into PDDL actions. PROHOW axioms are in the form $\alpha \Longleftarrow \beta$, where α and β are first order logic formula. Sequent calculus provides one way to reason with these formula to infer new ones by proving them true. In the sequent calculus, α can be said to be the *consequent* that can be inferred from β, the *antecedent*.

To translate the model into a PDDL domain definition each axiom $\alpha \Longleftarrow \beta$ of the model is translated into a PDDL action having the antecedent formula β as the PDDL precondition and the consequent formula α as the PDDL effect. The name, parameters and types of the action are defined accordingly, along with the types, requirements and predicates of the domain. For example, the following PDDL action definition represents Formula 11:

Listing 4.10: A PDDL action representing Formula 11.

```
(:action infer_finished
  :parameters (?e - environment)
  :precondition (exists (?g)
    (and
      (has_goal ?e ?g)
      (complete ?g ?e)))
  :effect (finished ?e))
```

The PDDL domain defined so far describes what can be inferred in the PROHOW model. However, it does not describe how an agent can modify the state of the world, adding new statements that could not have been inferred just by the model itself. An agent wanting to complete a checklist is allowed to do two actions. The first one is accomplishing a task, and it is identified by action accomplish_task(t,e,i). If t is a task ready in environment e, and execution i is an execution variable not previously assigned to any environment or task, then this action states that the execution variable i is assigned to task t and environment e and that it has succeeded.

The second action that an agent can perform is to define a new environment e as a sub-environment of a in which to complete task t through its method m. This action is identified with the name istantiate_sub_env(e,a,t,m). If m is a method of t, and e is a new environment variable not associated with any goal or any super-environments and different from a, then performing this action instantiates the environment e as a sub-environment of a and with goal t.

These two actions are sufficient to allow a planner to generate a valid checklist (if any exists) to achieve a certain goal. A valid checklist, is a partially ordered set of tasks, such that (1) the execution of its tasks in a sequence which respects its partial order does not violate the dependency constraints of its tasks and (2) the execution of all or a subset of its tasks is enough to infer the completion of its goal.

For example, we consider can consider the PROHOW representation illustrated in Figure 4.18. This diagram shows a task t which requires task r and which is decomposable into steps a and b, the second of which is decomposable by method x. We can represent the task of planning a checklist for task t with the PDDL problem definition listed in Listing B.2 in the Appendix. This problem definition defines the relations between the tasks. To set the goal of the problem, an environment e is defined as having goal t. The goal of the PDDL problem can now be stated as the goal of reaching a state where environment e is finished. Since in PDDL the creation of new objects is not allowed, a sufficient number of place-holder object variables should be reserved to allow the planner to instantiate new executions and environments.

The PDDL domain and problem definitions are sufficient to run an automated planner to try to discover possible solutions to this problem. Solutions to this problem detail the set of tasks that can be accomplished to accomplish a certain goal task, also specifying their order, the environments they have to be executed in, and the relations between such environments. For this reason, a solution to a PDDL problem generated from a PROHOW description of a set of instructions can be considered as an answer to the competency question Q8.

For the purpose of this evaluation, I have used the Fast-Downward planner[15] using

[15] http://www.fast-downward.org/ (accessed on 17/10/2017)

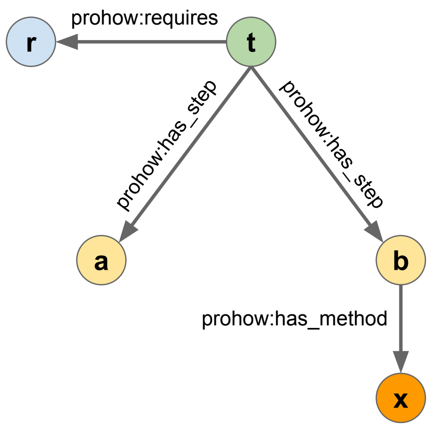


Figure 4.18: A simple model of instructions in the PROHOW model.

the “LAMA 2011 configuration”. The solution found for this problem is shown in Listing B.3. This solution can be interpreted as follows: task r is accomplished in environment e to make task t ready for execution, and then t is accomplished, thus completing the goal of checklist e. In this planning configuration, task a, b and x are redundant as t can be accomplished directly without resorting to decomposition.

In order to force the planner to decompose tasks, the *primitive tasks* extension can be introduced. Under this extension, the action accomplish_task(t,e,i) has a new precondition that makes it applicable only for tasks t defined as “primitive”. Intuitively, primitive tasks are tasks which are simple enough so that an agent can perform them without requiring further instructions. The problem description can now be extended with the assertion that only tasks r, a and x are to be considered primitive. This new configuration invalidates the previously found plan, as task t cannot be accomplished directly.

The new plan generated by the planner for this new configuration is shown in Listing B.4. This time, the planner decides to instantiate environment ee as a sub-environment of e to accomplish task b through its method x. The planner then proceeds to accomplish r in environment e, x in sub-environment ee and a in environment e. Having completed method x in sub-environment ee, the planner can now infer that task b has been accomplished in the super-environment e. Now that both steps a and b have been completed, the planner can infer that the task t has been completed and that the goal has been reached. More examples of how to define and solve planning problems containing bindings and constants can be found in Section B.1 of the appendix.

4.10 Domain Formalisation Evaluation

I now I validate the hypothesis that all the competency questions defined in Section 4.7.3 are answerable by the PROHOW model (Hypothesis 1). I do so by following the methodology described in Section 3.1.1. This methodology involves the implementation and testing of a concrete computational approach to answer the competency questions. Given that the expressiveness of the PROHOW model and the PROHOW vocabulary are equivalent, this hypothesis can be validated by demonstrating that the competency questions are answerable by data following the PROHOW vocabulary.

When using the PROHOW vocabulary, data is assumed to be stored in an RDF database that can queried using SPARQL. The SPARQL queries that are used in this evaluation are listed in Sections C.2 and C.3 in the Appendix. In some cases, executing one of these queries is enough to answer a competency question. For example, the following SPARQL query, based on the query template in Listing C.12 in the Appendix, returns the URIs of all the tasks that are complete in environment `http://example.com/env`, thus answering the competency question `Q4`.

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT ?task
WHERE {
  ?ex prohow:has_environment <http://example.com/env> .
  ?ex prohow:has_task ?task .
  ?ex prohow:has_result prohow:complete . }
```

When a single SPARQL query is not sufficient, complex functions are defined as algorithms that combine the results of one or more SPARQL queries. These algorithms are listed in Appendix D. The algorithms and queries that answer competency questions `Q1` to `Q8`, along with their typed parameters, are the following:

Q1 : `get_requirement_sets`(*task x*) returns a set of sets of tasks;

Q2 : `get_decomposition_sets`(*task x*) returns a set of sets of tasks;

Q3 : `get_tasks_in_env`(*environment e*) returns a set of tasks;

Q4 : `get_completed_tasks_in_env`(*environment e*) returns a set of tasks;

Q5 : `get_ready_tasks_in_env`(*environment e*) returns a set of tasks;

Q6 : `get_sub_environments_in_env`(*environment e*) returns a set of pairs of tasks and environments;

Q7 : `get_next_tasks_to_accomplish`(*environment e*) returns a set of tasks;

Q8 : `create_checklists`(*task x*) returns a set of lists of sets of tasks.

To evaluate these algorithms, I have implemented them in Python and I have tested them on a number of test cases. The test cases used for this evaluation are based on two sets of instructions and on four checklists, which represent these instructions at different stages of execution. These test cases have been designed to test all the axioms of the PROHOW model. To do so, the test cases include, among others, the following features:

- Tasks which do not have requirements.
- Tasks which have multiple necessary requirements (cumulative dependencies).
- Tasks which have multiple alternative requirements (sufficient dependencies).
- Tasks which have both necessary and alternative requirements (both necessary and sufficient dependencies).
- Tasks which do not have steps nor methods.
- Tasks which can be decomposed by steps.
- Tasks which can be decomposed by methods.
- Tasks which can be decomposed both by methods and by steps.

Based on these test cases, two types of tests have been conducted. The first type of test involves comparing the answers produced by the algorithms with a manually created set of expected answers. This type of test was used to directly verify the output of each algorithm.

The second type of test involves validating multiple competency questions at the same time by simulating the execution of a set of instructions. This simulation is performed as follows. Firstly, a checklist e to achieve a goal task x is generated using the algorithm to answer question `Q8`. Then, the checklist e is materialised as a set of RDF statements and added to the local knowledge base. Lastly, the tasks of checklist e are selected, one at a time, to be executed. The order in which tasks are selected follows the ordering imposed by the checklist. After a task is selected, its successful execution is simulated by adding a statement to the knowledge base that marks it as complete in that environment.

The test is considered failed if one of two fail criteria occurs. The first fail criterion occurs if the execution of a task is about to be simulated, but the task is not ready in that environment yet. This criterion validates that the partial ordering imposed by the checklist is consistent with the requirements of each task.

The second fail criterion occurs if the execution of all the tasks of the checklist have been simulated, but the goal task x cannot yet be inferred to be complete. This criterion validates that the successful completion of the checklist is sufficient to achieve its goal. If no fail criteria occurs, then the test is considered passed.

The code and data for this experiment is available on GitHub[16] and it uses the RDFLib[17] library to read and modify RDF data from a Turtle file. In this implementation, all the tests for each of the algorithms validate successfully, thus providing evidence in favour of the correctness of Hypothesis 1.

4.11 Conclusion

In this chapter I have followed the NeOn method for ontology engineering to formalise the domain of step-by-step instructions and their execution as to-do checklists. This method resulted in PROHOW, a logical model of this domain, along with a concrete implementation of this model in RDF. This formalisation is the solution to the know-how representation problem that I will consider throughout the rest of this thesis.

Existing languages typically assume that processes are composed of atomic building blocks. For example, *primitive activities* represent the most fine-grained level of task decomposition in HTN planning. In a similar way, individual web services, such as the ones defined by OWL-S, can be seen as the atomic components of web service composition workflows, which cannot be decomposed further into other services or workflows. The PROHOW model, instead, is designed to model the open-ended domain of human instructions. In PROHOW, tasks do not need to be defined in exact terms, and they can always be decomposed into more fine-grained tasks or alternative methods.

This formalisation was driven by a number of competency questions, which cor-

[16] `https://github.com/paolo7/prohow_algorithms` (accessed on 17/10/2017)
[17] `https://github.com/RDFLib` (accessed on 17/10/2017)

respond to the questions that this formalisation should be able to answer to support a user in performing two key applications. The first one is discovering relevant know-how when exploring sets of instructions. The second is obtaining information about the current state of execution of a to-do checklist and discovering how to progress it toward the completion of a certain goal. A computational solution to these competency questions was provided by a number of algorithms and database queries, which has been implemented and tested.

It should be noted that the core of the PROHOW model is a set of three disjoint concepts: *tasks*, *executions* and *environments*. Unlike typical ontologies, PROHOW does not arrange concepts and properties hierarchically into a taxonomy. For this reason, PROHOW can be seen more like a collection of semantic terms, or a vocabulary, than an ontology. The lack of a taxonomy is an important difference, which significantly reduces the applicability of existing ontology tools and methods when dealing with PROHOW. For example, the OntoClean methodology for ontology evaluation developed by Guarino and Welty (2009) is not applicable to PROHOW as it is strongly focussed on taxonomic relations.

As part of the NeOn method, related ontological and non-ontological sources, such as process description languages and instructional websites, were analysed and incorporated in the formalisation as far as possible. This integration of resources is meant to make the formalisation compatible both with formal languages and with informal instructional resources. The model generated by this method was translated into the Automated Planning language PDDL to show its compatibility with existing formal languages. Its compatibility with informal instructions, instead, will be validated in the following chapter, which focuses on the automatic extraction of PROHOW instruction descriptions from real-world instructional websites.

Chapter 5

Know-How Acquisition

Procedural knowledge can be found on the web in many different formats. For example, instructions are commonly found in a textual format, such as those found on the wikiHow website,[1] or in a visual and audio format through instructional videos, such as those found on the Howcast website.[2] Other forms of procedural knowledge can also be found in a non-instructional format, such as in Question and Answering (Q&A) websites or in social media posts. Human procedural knowledge on the web covers a wide range of subjects, such as cooking recipes,[3] technology hacks,[4] and business know-how.[5]

This variety of formats has led to the development of different approaches to extract procedural knowledge. In this chapter I will demonstrate how to extract procedural knowledge from semi-structured textual step-by step instructions. The notion of semi-structured resources used in this thesis refers to the notion of HTML documents conforming to certain templates (Álvarez et al. 2008). This type of instructions has been chosen because it is a class of resources that is both common and easy to process. My proposed approach is complementary to existing pieces of work, which I have reviewed in Section 2.3, on extracting procedural knowledge from a wider range of sources using probabilistic approaches. Although more widely applicable, those approaches suffer from limitations in precision and recall. These limitations are particularly problematic during the execution of instructions, and especially when this involves collaboration, since agents might need to reason about the state of the execu-

[1]`http://www.wikihow.com/` (accessed on 17/10/2017)

[3]`https://www.wikihow.com/Make-Pancakes` (accessed on 17/10/2017)

[4]`https://snapguide.com/guides/turn-your-smart-phone-into-a-wi-fi-hotspot/` (accessed on 17/10/2017)

[5]`https://www.wikihow.com/Create-an-Investment-Plan` (accessed on 17/10/2017)

tion of a task in exact terms in order to achieve coordination.

This chapter is organised as follows. I define a specific knowledge-acquisition problem in Section 5.1, and I propose a method to address it in Section 5.2. I evaluate this method with a set of experiments (Section 5.3), describe its results (Section 5.4), and assess the extent to which they solve the original problem (Section 5.5). In Section 5.6 I demonstrate how the core principles of this method can also be used to acquire know-how directly from users. I then conclude the chapter with a summary of its main findings in Section 5.7.

5.1 Problem Description

The problem addressed in this section is the automatic extraction of procedural knowledge from human-made textual instructions. In an effort to address this problem, the following hypothesis is formulated:

Hypothesis 3. *The proposed knowledge extraction method applied to a standard semi-structured textual representation of instructions automatically generates a* complete *and* correct *procedural knowledge representation of those instructions in the* PROHOW *model.*

The extraction method proposed in this thesis will be described in Section 5.1.1. A set of instructions is said to be *semi-structured* if two semantic relations, namely the dependency between tasks, and their decomposition into sub-tasks, are represented by certain structural (i.e. non-lexical) properties, such as the relative position of two pieces of text in a document. If these structural properties carry the same meaning across all of the documents contained in a website, then this website is said to contain textual representations of instructions that are semi-structured in a standard way.

Following the methodology presented in Section 3.2.1, I define the usage of the terms "correct" and "complete" in Hypothesis 3 with respect to the competency questions defined in Section 4.7.3 for instructional articles, namely `Q1` and `Q2`:

Complete The PROHOW representation P of a set of instructions I is called *complete* if for every answer a of a competency question q that human judgement would generate, a could also be inferred automatically as a correct answer for question q over representation P.

Correct The PROHOW representation P of a set of instructions I is called *correct* if for every answer a computed on the representation P for a competency question q, human judgement would accept a as an answer of question q for instructions I.

To avoid additional complications of interpreting human judgement, I assume that a human expresses it by selecting a number of items in a set of options generated from the available resources. For example, a human could select in a list of task labels the ones that he or she believes represent the steps of another task. Options that do not appear textually in the set of instructional resources are not considered candidate for human judgement. This approach for computing human judgements can be used to verify or falsify the given hypothesis.

One important assumption made in this thesis is that the original sets of instructions are correct themselves. In this thesis this will be called the *sic* assumption (from Latin *sic erat scriptum*, meaning “thus was it written”). Under this assumption, no effort is made to validate or rectify the original sources. Due to the natural fallibility of human generated content, errors can in fact be found in the original sources of the instructions. These errors can be grouped into two categories. *Semantic errors* occur in the human-understandable description of an entity. For example, a step of a cooking recipe that should communicate that the oven needs to be preheated to 180 degrees can be said to be semantically incorrect if its textual label reads as follows: “preheat the oven to 200 degrees”. *Structural errors* are errors that affect the relations between the various components of a set of instructions. For examples, if the step about preheating the oven is defined as a method instead of a step, then it is said to be structurally incorrect. If errors exist in the original sources, the proposed method should generate a faithful representation of the erroneous source.

5.1.1 Know-How Acquisition from Semi-Structured Resources

One of the advantages of a simple vocabulary that closely models how humans describes procedures, as discussed in Section 4.5, is the ability to easily extract knowledge from human-generated instructions. Explicit representations of PROHOW-like concepts are not rare in such instructions. For example, websites like wikiHow explicitly subdivide instructions into an ordered set of steps and clearly identify the requirements for such instructions. In the context of this thesis, therefore instructional resources contained in wikiHow can be said to be semi-structured. It is important to note that this structure is not the result of a dedicated annotation process, but it is instead spontaneously created by humans for humans. A more structured representation of a process, in fact, is not only more understandable by machines, but first and foremost by people. This structure is considered good practice as it makes the overall description of the process clear and reduces ambiguities. In fact, many websites like wikiHow, Snapguide and Howcast do not only recommend such practice, but require it for all of their instructional pages.

As a concrete example, we can consider the following set of instructions in both unstructured and structured format which arguably contain the same semantic content:

Listing 5.1: A pancake recipe as an unstructured set of instructions.

```
This set of instructions will tell you how to make a pancake. You
will need eggs, milk and flour. First of all, prepare the mix. Then
pour the mix in a hot pan. Finally cook it until golden.
```

Listing 5.2: A pancake recipe as a semi-structured set of instructions.

```
How to Make a Pancake.
Steps:
1. Prepare the mix
2. Pour the mix in a hot pan
3. Cook it until golden
Requirements:
- Eggs
- Milk
- Flour
```

Apart from the likely benefits to human understanding, semi-structured resources are easier for machines to parse. For example, the question “how many steps does this set of instructions have?” can easily be answered by counting the number of bullet-points that immediately follow the “Steps” keyword. Similarly, the question “which is the first step of the instructions?” can be answered by looking at the text in the first of such bullet-points.

5.2 A Method for Know-How Acquisition

Exploiting the fact that instructions found within a particular website have a common template, it is possible to create a rule-based knowledge extractor. This extractor would apply certain rules to segment the page into components and detect their semantic role.

For example, it might be the case that all the instruction pages of a website (and only those) contain one HTML ordered list with label “Steps”, such that each element in the list corresponds to a step of the instructions. In this scenario, an extractor could easily identify these lists and extract from them the list of steps. Extractors could also be used to exploit templates that are common throughout a website even though some pages do not follow them. Web pages that do not adhere to these templates, however, might not be parsed correctly.

Rules to extract the various components of a set of instructions from an web page can be defined using the Cascading Style Sheets (CSS) *selectors* (Hickson et al. 2011). CSS selectors define a syntax to identify specific elements within an HTML file. For example, the CSS selector “`div.steps`” identifies all of the `div` tags which belong to the class `steps`. These rules are specific to a particular template that is found within a website. For this reason, a new set of rules might be required if the template of the website changes, or if we want to apply this method to a different website. Given that changes in the structure of a website tend to be infrequent, I argue that this is not a severe limitation to my approach.

As an example, I will now list five rules that can currently[6] be used to extract the components of a set of instructions from wikiHow. I use the notation $\sigma(x)$ to refer to the process of returning the set of HTML elements which match the CSS selector x.

Main Task : $\sigma(\texttt{h1})$

Task Set : $\sigma(\texttt{h3} + \texttt{div} > \texttt{ol})$

Task Set Type : $\sigma(\texttt{div.steps} > \texttt{h3} > \texttt{div})$

Inner Sub-Steps : $\sigma(\texttt{li})$

Requirements : $\sigma(\texttt{div.ingredients} > \texttt{h2} + \texttt{div} > \texttt{ul} > \texttt{li})$

The element identified by the *Main Task* rule contains label of the main task t described in the set of instructions. Rule *Requirements* identifies the labels of the set of requirements R of the set of instructions ($\forall r \in R.\texttt{requires}(t,r)$). Rule *Task Set* identifies the labels of a set of tasks X which immediately decomposes t. If the elements identified with *Task Set Type* contain the textual string “Method”, then X represents a set of alternative methods ($\forall x \in X.\texttt{has\_method}(t,x)$). If they contain the textual string “Part”,

[6]Tested on the 28th of March 2017.

then X represents a set of necessary parts ($\forall x \in X.\texttt{has\_step}(t,x)$). Rule *Inner Sub-Steps* is then applied to the inner-HTML of each method and part x to extract the labels of their steps Y ($\forall y \in Y.\texttt{has\_step}(x,y)$). The order by which the steps are found in the file dictates how they should be ordered in the PROHOW representation. These rules extract the HTML code snippet that generates the human-readable textual description of each component of the set of instructions.

Once these components have been identified, they can be stored in RDF. A website-specific procedural knowledge extractor can be seen as a system that takes a web page from a specific website as an input, and outputs an RDF representation of the instructions present in the page, if any. To preserve provenance, the Web Annotation Vocabulary (Young et al. 2017) is used. This vocabulary is used to formalise an *annotation* object that relates the RDF description of a task (called *body* of the annotation) to the specific portion of the original web resource from which it was extracted (called *target* of the annotation). This annotation object can be enriched with further metadata, such as information about when the extraction was performed, and by which system.

If the template used by an extractor is found consistently throughout a website, then all the instructions within the website can be extracted. This ensures that the extracted knowledge base does not suffer from a loss of recall. Moreover, if the template clearly identifies the components of a set of instructions, then the extracted knowledge might not suffer from a lack in precision.

An effective use of such an extractor is in combination with a *web crawler*, also known as "spider". A web crawler is software capable of automatically exploring (crawling) web pages by following the HTML links found in the source code of the pages. Web crawlers start their crawl with a non-empty list of URLs to visit. The list of URLs that a crawler still needs to visit is called "frontier", while the URLs that populate this list at the beginning of a crawl are called *seed* URLs.

To extract the RDF representation of all the instructions found in a website, a crawler can be initialised with one or more seed URLs pointing to pages from that website, such as its main page. In practice, it is important to limit the crawl to pages within the domain of the website, or otherwise the crawler will attempt to crawl all of the web resources which are reachable by following links; a task that is not likely to terminate within an acceptable time-frame. This can be simply done by allowing only URLs from the website's domain into the crawl frontier. Once a page is retrieved, its source code will be fed to the extractor that will generate its RDF representation. The URIs of the entities discovered are minted as described in Section 2.1.1. It should be

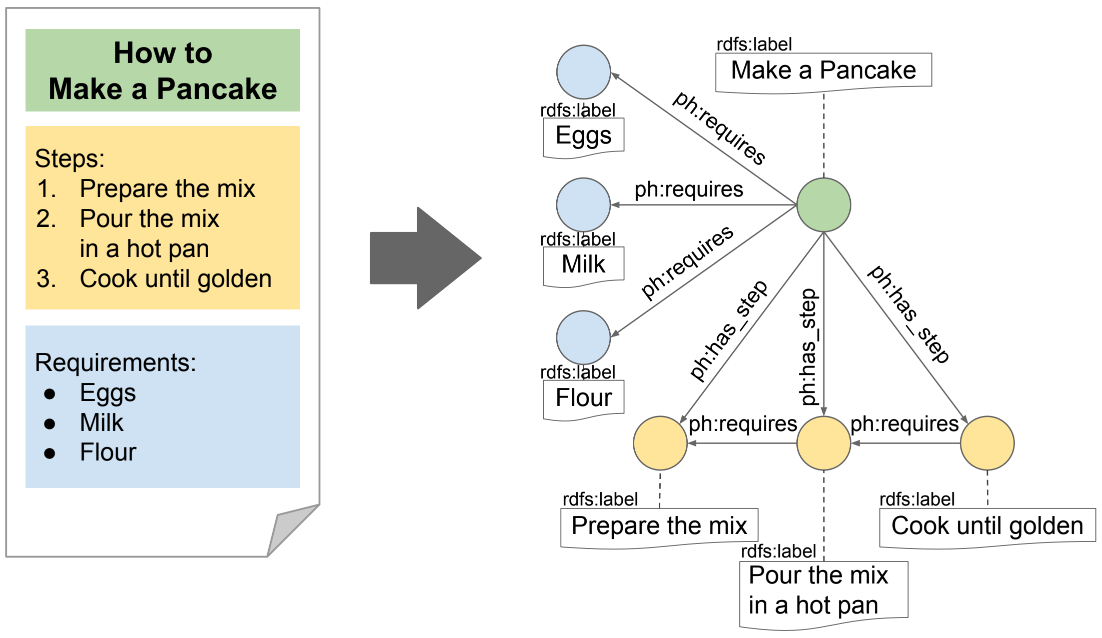


Figure 5.1: Extraction of a PROHOW RDF representation from a semi-structured set of instructions. To enhance readability, the RDF namespace `prohow:` has been shortened to `ph:`.

noted that new URIs are minted for every run of the extraction. This has the advantage that if an entity has changed between the time when two extractions are performed, then each version of this resource will have its unique URI. For example, the step of a process might initially have label: "Preheat the oven to 180 degrees", which is later changed to "Preheat the oven to 200 degrees". These two different versions of the same step are arguably not equivalent, and this is reflected by assigning different URIs to them. The drawback of this approach, however, is that it can generate multiple URI synonyms for the same entity, if this entity does not change between extractions.

5.3 Experiments in Knowledge Extraction

I have tested this approach to knowledge extraction on the wikiHow and Snapguide websites (Pareti et al. 2014a,b). The wikiHow website is a community-driven effort to create a high-quality and comprehensive repository of human know-how. Similarly to Wikipedia,[7] each set of instructions within wikiHow can be edited by any user, making the curation of its content a communal responsibility. This fact generally leads to a consistently high quality of the articles, but at the same time to a high level of

[7]`http://en.wikipedia.org` (accessed on 17/10/2017)

standardisation of the instructions. The policy of wikiHow, in fact, is to favour the integration of all the know-how about a certain task within a single set of instructions, and to define different methods only when this is strongly required.

This is in stark contrast to Snapguide, a know-how repository mainly targeted at mobile users. Within Snapguide, sets of instructions belong to individual users, which are the only ones allowed to edit them. It can be observed that this leads to a much richer variety of solutions to complete the same task, as each user is allowed to describe their particular approach to achieve a goal. However, without benefiting from a community effort, the quality of the articles can vary greatly. While one Snapguide user might write excellent instructions, another one might write intentionally misleading ones. In several cases, in fact, I have observed steps with an unintelligible description or with no description at all.

The main reason behind my choice of these websites is that they contain semi-structured procedures that adhere to a standard template. For both websites, the template includes an ordered list of steps and a set of requirements. In the case of wikiHow, this template also includes an optional subdivision of steps into different methods, such as different ways to prepare a pancake, or an optional further decomposition of steps into sub-steps. In the English version of wikiHow, methods were identified when different lists of steps were grouped under the "Method" keyword. Multiple lists of sub-steps, instead, were identified when grouped under the keyword "Part". The PROHOW vocabulary can capture this semantic distinction by connecting the main goal to each of the lists with the relation `prohow:has_method` in the former case, and `prohow:has_step` in the latter.

Other reasons behind this choice of websites are the following:

- They are leading examples of instructional websites, having among the largest number of articles and users. This was shown in Table 4.1 in the previous chapter.

- They are domain independent. Instructions of any type can be added to the website, whether it is related to cooking, dating, technology or any other field.

- Their differences provide evidence of the generality of this method. For example, one of them is community-driven, while the other is user-driven; one of them targets generic web users while the other focuses on mobile users.

- Their terms of service at the time when this experiment was performed, namely in May 2014, were compatible with this study. The content uploaded on wikiHow, for example, is free to modify, republish and share under the Attribution-Noncommercial-Share Alike 3.0 Creative Commons License.[8] Snapguide, at the time this experiment was performed, did not reuse a standard licence. However, its terms and conditions were compatible with this experiment.

While this knowledge extraction focused on extracting PROHOW relations between entities, more information about a set of instructions can be usually found in a web document. For example, a website might publish the number of views of a set of instructions, its rating, or some other measure of its quality. While these pieces of information can be seen as website dependent, a common type of additional information comes from the *category* of a set of instructions. Websites like wikiHow and Snapguide classify their sets of instructions under a certain number of categories. For example, instructions on "how to make a pancake" could fall under the *recipes* category, while the one on "how to apply for a job" could fall under the *employment* category. The two category structures of wikiHow and Snapguide were extracted and and represented as two hierarchies of `rdfs:Class` instances connected with `rdfs:subClassOf` relations. These category structures are prominently displayed in the wikiHow and Snapguide websites. Information on the exact category structures that were found when this study was made has been published online.[9]

To ensure interoperability, the two hierarchies were manually integrated into a unified hierarchy. It was possible to perform this integration manually as Snapguide's hierarchy is a flat set of 17 different categories, which could be individually linked with their closest wikiHow counterpart. For example, the Snapguide category for *food* was labelled as a subclass of the wikiHow category for *food and entertainment*. Overall, the unified hierarchy consists of 2759 categories linked together by 2742 subclass relations.

5.4 Knowledge Extraction Results

The results of this knowledge extraction are listed in Table 5.1. Overall, more than $200,000$ sets of instructions were analysed and decomposed into over 2.5 million labelled entities. These statistics cannot be compared with official figures as up-to-date

[8]`http://creativecommons.org/licenses/by-nc-sa/3.0/` (accessed on 17/10/2017)
[9]`https://github.com/paolo7/KnowHowDataset` (accessed on 17/10/2017)
[9]`http://www.howcast.com/` (accessed on 17/10/2017)

	wikiHow	Snapguide	Total
Number of instruction articles (C.1)	166,664	44,464	211,128
Requirements of main processes (C.3)	335,962	194,390	530,352
Main processes with requirements	59,166 (35.5%)	34,925 (78.5%)	94,091 (44.5%)
Steps (C.4)	1,361,169	498,914	1,860,083
Main processes with steps	166,518 (99.9%)	42,117 (94.7%)	208,635 (98.8%)
Methods (C.5)	56,170	0	56,170
Main processes with methods	18,295 (10.9%)	0 (0%)	18,295 (8.6%)
Total number of labelled entities (C.6)	1,920,575	737,768	2,658,343

Table 5.1: Statistics computed on the HOWDB dataset generated in May 2014. The reference in brackets refers to the related SPARQL queries listed in Appendix C. These are not official figures reported by the websites themselves.

official figures were not available at the time the experiment was run. As described in the preceding section, these instructions come from wikiHow and Snapguide. This dataset, that will be referred to as HOWDB, has been published on the GitHub[10] and Datahub[11] websites. The unified hierarchical structure of the wikiHow and Snapguide websites has also been published along with this dataset. The sections of HOWDB containing wikiHow and Snapguide articles will be called, respectively, HOWDB-wikiHow and HOWDB-Snapguide.

Table 5.1 also lists the number of requirements, steps and methods that were found, along with the number of set of instructions that contain these types. As it can be expected, almost all of the dataset (98.8%) is composed by tasks which are decomposable with steps. Instructions with requirements are very common, but more frequent in Snapguide (78.5%) than in wikiHow (35.5%). This could be a result of a higher focus of Snapguide in the cooking and crafts domain, which typically require a set of ingredients and tools. Sets of instructions on social know-how, such as "How to Communicate with a Deaf Person", are instead popular on wikiHow, and they often do not have specific requirements. Figures 5.2, 5.3 and 5.4 show, respectively, the distribution of the number of steps, requirements and methods across the set of instructions. Figures 5.2 and 5.3 compare the whole dataset of all the sets of instructions with its Snapguide and wikiHow partitions. These figures show a very similar distribution of the number of requirements across the two websites. This similarity is also

[10]`https://github.com/paolo7/KnowHowDataset` (accessed on 17/10/2017)
[11]`https://w3id.org/knowhow/dataset` (accessed on 17/10/2017)

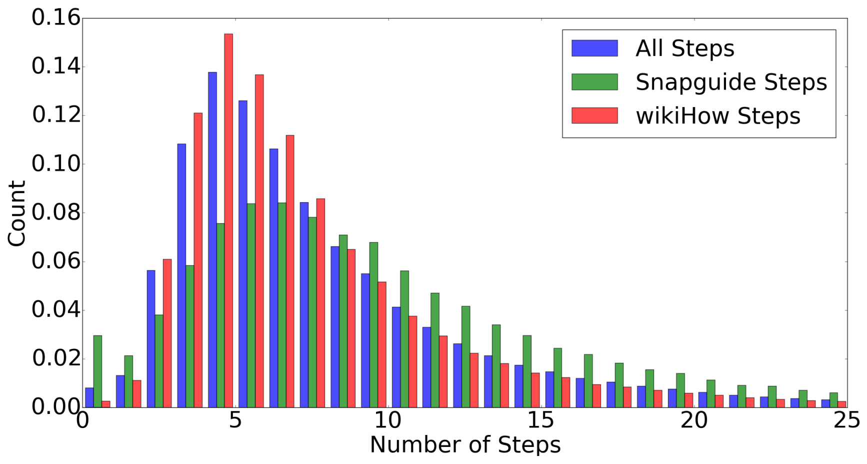


Figure 5.2: Normalised histogram of the number of steps of the instructions from wikiHow, Snapguide and their union. Only sets of instructions with at least one step are considered.

reflected by the average number of requirements per set of instructions, namely 5.6 for wikiHow and 5.5 for Snapguide. The number of steps, on the other hand, follows a slightly different distribution, with wikiHow instructions having a lower average (8.1) and standard deviation (5.5) than the average (11.8) and standard deviation (7.7) of Snapguide instructions. This difference could be attributed to to wikiHow's community editing which results in a larger degree of standardisation, such as a more standard and self-contained number of steps per article. Methods appear in about one in ten sets of wikiHow instructions. Each set of instructions with methods has an average of 3 methods, and their distribution is shown in Figure 5.4.

These results demonstrate the existence of a large class of instructional resources that can be automatically converted into a structured representation. More specifically, these results can be seen as evidence that semi-structured instructional websites are a significant portion of online know-how, and they can be turned into an RDF representation using the PROHOW vocabulary. Although the experiments were conducted only on two websites, it can be assumed that this method would lead to similar results on other semi-structured instructional websites such as Instructables[12] or BBC Recipes.[13]

[12]`http://www.instructables.com` (accessed on 17/10/2017)
[13]`http://www.bbc.co.uk/food/recipes/` (accessed on 17/10/2017)

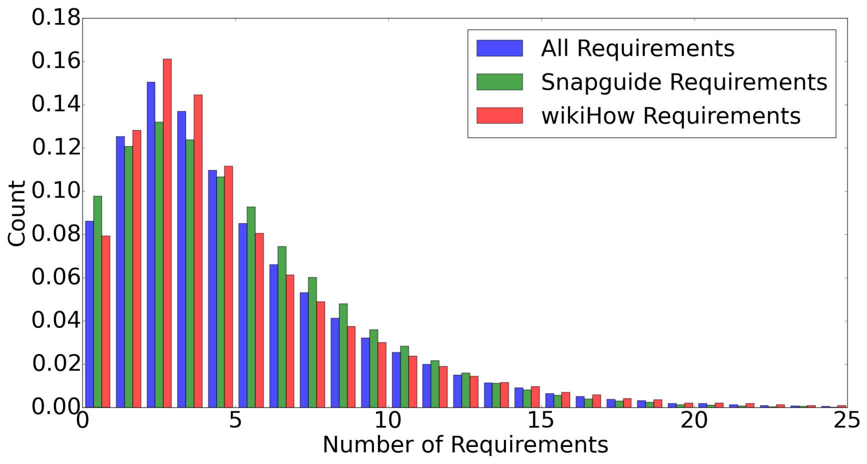


Figure 5.3: Normalised histogram of the number of requirements of the instructions from wikiHow, Snapguide and their union. Only sets of instructions with at least one step are considered.

5.4.1 Multilingual Know-How Extraction

One of the advantages of structure-based knowledge acquisition is language independence. In wikiHow, instructional articles have the same structure across different languages. It is therefore trivial to extend the proposed knowledge extraction approach to languages other than English. To demonstrate this point, a crawl of wikiHow has been performed over 15 different languages. The results of this knowledge extraction are detailed in table 5.2 and the extracted RDF data has been published on the data-science platform Kaggle.[14] This dataset containing multilingual instructions from wikiHow will be called HOWDB-Multilingual.

It should be emphasised that the data discussed in this thesis refers only to the HOWDB dataset, which was obtained from a crawl of those websites performed in May 2014, and restricted to English only. While this data accurately describes properties of instructions on these websites at that particular time, these properties are known to have later changed. For example, the number of know-how articles in these repositories has been steadily growing larger over time.

[14]`https://www.kaggle.com/paolop/human-instructions-multilingual-wikihow` (accessed on 17/10/2017)

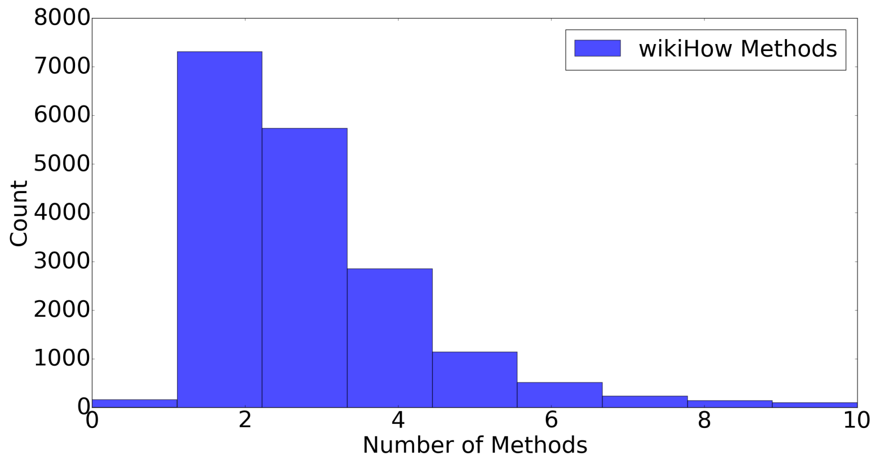


Figure 5.4: Histogram of the number of methods of the instructions with at least one method.

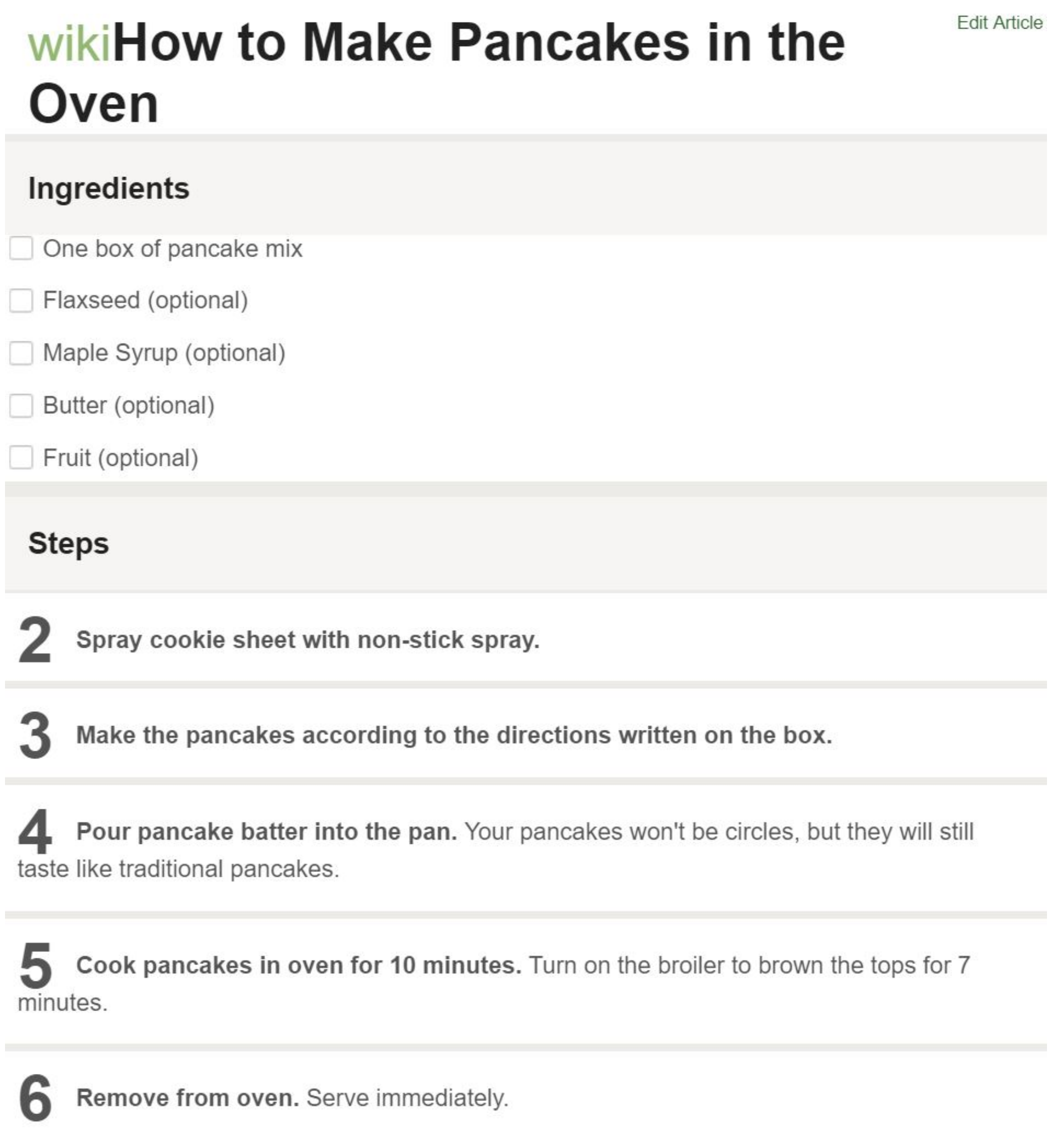
Language	Number of Articles	Language	Number of Articles
Russian	127,738	Dutch	19,318
Spanish	120,507	Arabic	15,589
Portuguese	92,520	Czech	10,619
Chinese	82,558	Thai	10,213
Italian	79,656	Vietnamese	8,670
French	60,105	Korean	7,606
German	57,533	Hindi	6,519
Indonesian	39,246		

Table 5.2: Statistics of the multilingual extraction performed in February 2017.

5.5 Analysis of the Extracted Data

Are the extracted PROHOW instruction representations correct and complete? Following the methodology outlined in Section 3.2.1 we can address this question by considering the competency question previously set for this type of data, namely `Q1` and `Q2`. In particular, human judgements can be collected to determine whether human answers to these questions would differ depending on whether humans are exposed to the original instructional webpages, or to a visualisation of the knowledge extracted from them.

I argue that such experiment is not required, as the extracted knowledge is equivalent to the one contained in the source webpages with respect to the competency



wikiHow to Make Pancakes in the Oven

Edit Article

Ingredients

- ☐ One box of pancake mix
- ☐ Flaxseed (optional)
- ☐ Maple Syrup (optional)
- ☐ Butter (optional)
- ☐ Fruit (optional)

Steps

2 **Spray cookie sheet with non-stick spray.**

3 **Make the pancakes according to the directions written on the box.**

4 **Pour pancake batter into the pan.** Your pancakes won't be circles, but they will still taste like traditional pancakes.

5 **Cook pancakes in oven for 10 minutes.** Turn on the broiler to brown the tops for 7 minutes.

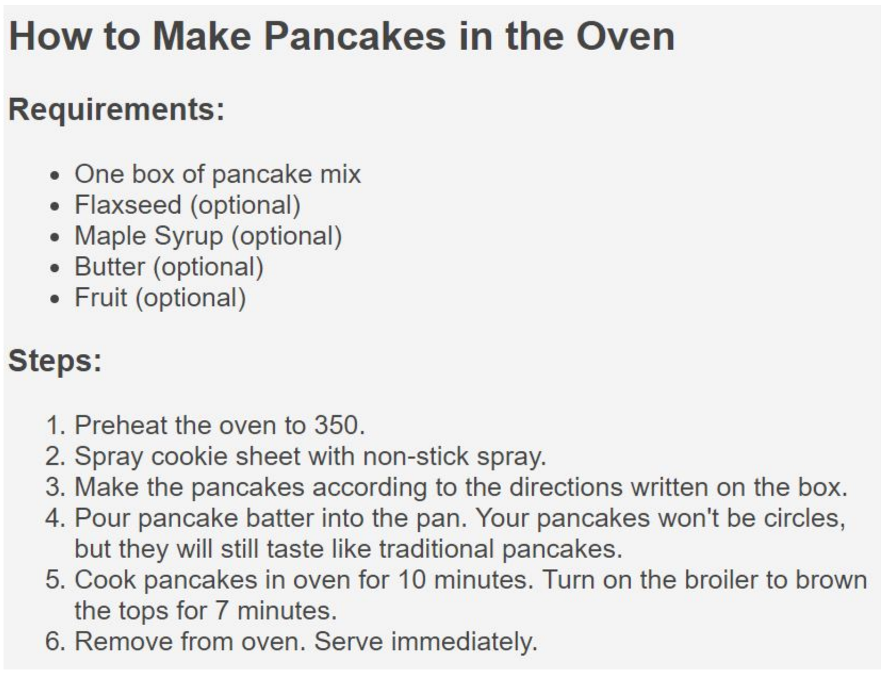
6 **Remove from oven.** Serve immediately.

Figure 5.5: Screenshot of a visualisation of the set of instructions on "How to Make Pancakes in the Oven" published on wikiHow.

questions. In fact, the knowledge extraction system I proposed extracts a knowledge model which directly maps to the structure of the original instructional webpages. As a result of this, the extracted knowledge can be visualised in an equivalent way as in in the original sources.

For example, we can consider the set of instruction on "How to Make Pancakes in the Oven" available on wikiHow.[15] Figure 5.5 displays a screenshot of a browser visualisation of this set of instructions. It should be noted that this visualisation has been edited by removing the pictures that were originally displayed along with each step

[15] `http://www.wikihow.com/Make-Pancakes-in-the-Oven` (accessed on 17/10/2017)



How to Make Pancakes in the Oven

Requirements:

- One box of pancake mix
- Flaxseed (optional)
- Maple Syrup (optional)
- Butter (optional)
- Fruit (optional)

Steps:

1. Preheat the oven to 350.
2. Spray cookie sheet with non-stick spray.
3. Make the pancakes according to the directions written on the box.
4. Pour pancake batter into the pan. Your pancakes won't be circles, but they will still taste like traditional pancakes.
5. Cook pancakes in oven for 10 minutes. Turn on the broiler to brown the tops for 7 minutes.
6. Remove from oven. Serve immediately.

Figure 5.6: Screenshot of a visualisation of the PROHOW data extracted from the set of instructions displayed in Figure 5.5.

of the set of instructions. In this work, I am assuming that wikiHow and Snapguide instructions are primarily textual. Images, instead, are assumed to be useful, but not necessary, to make sets of instructions more understandable. If this assumptions were found to be false, the same technique used to select textual description of the components of a set of instructions can be extended to detect and record images.

We can then compare Figure 5.5 with Figure 5.6, which displays the screenshot of a visualisation generated from the PROHOW data extracted from the same webpage. While these figures differ in some of their graphical details, the textual content can be arguably be considered equivalent. This equivalence is not specific to this particular set of instructions, and it can be generalised to the other sets of instructions.

More specifically, Question `Q1` asks what set of sets of requirements are needed by a task t. The vast majority of HOWDB instructions contain a single list of requirements. I assume that the human interpretation of this list is the set of entities needed by the task. In this case, under the sic-assumption, the list is an exhaustive and correct set of requirements needed by the task. Since all of the list items are collected by the crawler, and since their human-understandable representation is extracted without modification, it can be concluded that the extracted PROHOW representation of these requirements is also complete and correct.

Some of the articles in wikiHow, however, contain multiple lists of requirements. It is unclear in this case whether all of the requirements in all of the lists are needed, or whether satisfying only one of the lists is sufficient. In HOWDB-wikiHow, this multiple lists of requirements were found in $1,164$ articles (computed with SPARQL query C.7), a small portion of the overall dataset. If these articles are discarded, HOWDB-wikiHow would lose 0.7% recall. If they are included, then the final dataset would have up to a 0.7% loss of precision. For completeness purposes, the current version of HOWDB-wikiHow includes articles which have multiple sets of requirements and it makes the conservative assumption that all of them are needed. Although the current version of the dataset does not address this problem, approaches to determine the particular semantics of these lists can be easily imagined. If certain items are repeated across these lists, then it is likely that only one of the lists is needed. For example, a high degree of overlap can be expected between the list of requirements “Ingredients for the chocolate cake” and the list “Ingredients for the fruit cake”. Moreover, the name of the lists can often provide clues as to which part of the instructions they apply to. For example, it might be possible to discover that both lists of requirements “Ingredients for the sponge base” and “Ingredients for external decoration” are needed, if they map to two necessary parts of the instructions “Make the sponge base” and “Make the external decoration”.

The same argument made for competency question `Q1` can be extended to question `Q2`. All of the instructions in HOWDB-Snapguide and most of the instructions in HOWDB-wikiHow contain a single list of steps. I assume that the human interpretation of a list of steps of a super-task is the set of sub-tasks needed in order to complete the super-task. Under the sic-assumption, if the original list of steps is complete and correct, then so will be the extracted PROHOW representation. Some instructions in HOWDB-wikiHow contain multiple lists of steps, and it is therefore important to determine if all of them are needed, or if each list of steps represents one of multiple methods to complete the instructions. Fortunately, the PROHOW model can reliably capture this semantic distinction since the wikiHow website consistently uses two different terms, “Part” and “Method”, to explicitly discriminate between these two different scenarios. Therefore, the PROHOW representation of a set of instructions can be seen as semantically equivalent to the human interpretation of the same set of instructions with respect to competency question `Q2`.

It should be mentioned that certain steps and requirements of a set of instructions can be considered as “optional”. The concept of an optional task could be a useful extension of the PROHOW model, but I will argue now why it is not required for the scope of this project. For example, we can consider these three steps of a process t to make tea: a: "boil water", b "add tea leaves in the boiled water" and c: "optionally, add milk and sugar". Although step c can be considered an optional task, this does not change its semantic interpretation according to the PROHOW model, or in other words, that all three steps a, b and c need to be accomplished before t can be considered complete. In fact, it would not be correct to assume that an agent that has completed steps a and b has finished completing task t. For example, a waiter might follow this procedure to serve tea to a customer. Even though the customer might not want sugar and milk to be added in the tea, the waiter should acknowledge this step before assuming that the overall goal is complete.

By analogy with the mathematical difference between a model that does not contain a set, and a model that contains the empty set, the lack of an action should not be confused with the act of performing an *empty action*. The concept of empty actions is common in many process description languages. Intuitively, an empty action is an action that can be accomplished by an agent simply doing nothing, but it is an action that needs to be accomplished nonetheless. Within the PROHOW model, an optional task could be modelled as a task that can be marked as complete after completing an empty action.

The evaluation of these results shows how the formulated hypothesis is validated on the HOWDB-Snapguide dataset, and how it is validated on a specific subset of the HOWDB-wikiHow dataset amounting to 99.3% of the instructions extracted from wikiHow. This hypothesis has important practical implications. Given the availability of large domain-independent instructional websites, the validation of this hypothesis implies that large datasets of PROHOW procedures can already be created automatically. Moreover, the fact that such websites are populated by non-expert web users demonstrates that PROHOW representations of previously unknown sets of instructions can be acquired on demand by non-expert users, for example through the use of forms.

5.6 Know-How Acquisition from Forms

Knowledge extraction from a corpus of existing resources can help bootstrap the creation of a knowledge-base. When new knowledge is created, however, it can be more efficient to acquire it directly from the users. A typical way in which users can help a system to acquire knowledge is through the use of forms. Forms, such as HTML forms, can be seen as a list of fields. In a simple form, each field is characterised by a label, which explains to the user what type of input is required, and by an input field, which collects the input from the user and optionally formats it.

The large volume of existing semi-structured instructions can be taken as evidence that non-expert users are familiar with concepts such as "steps", "methods" and "requirements". It is therefore possible to create a usable form that acquires the same type of information contained in semi-structured instructions. I have created a prototype of this kinds of form as and it is available online as part of the KnowHow4j[16] tool. This tool is entirely written in Javascript and it can be used in any modern browser. It is composed of two functionalities, namely an editor and a visualiser. I will present now the editor functionality of this tool, while the visualiser functionality will be discussed later in Section 7.2. The KnowHow4j editor is a form to acquire PROHOW representations of semi-structured instructions provided by the user. This form contains a single text field, where users can list the description of a task, along with a list of steps, methods and requirements. Users are asked to follow a simple convention, namely to write each step, method or requirement in a new line, respectively prefixed with the keywords "step", "method" or "requirement".

After the user completes the form and clicks on the "Parse into RDF" button, their input is parsed into an RDF representation and then translated into an intuitive human-readable representation in HTML. This step allows the user to detect whether the machine's interpretation of the data is correct, as errors in the interpretation will result in an incorrect human-readable representation. When the user is satisfied with the result, he or she can retrieve the generated HTML code snippet of the instructions. This snippet can then be posted on a web page by the user, either manually or by using a web page creation tool which accepts HTML input. This code snippet offers a compact human readable representation of the instructions, along with the corresponding PROHOW RDF data encoded in the same HTML snippet using RDFa (Herman et al. 2015). RDFa is W3C recommendation that specifies a standard approach to embedding RDF data into HTML elements and allows HTML documents to be both human and machine readable.

The HTML and RDFa code generated by KnowHow4j demonstrates how the PROHOW structured representation of instructions can co-exist with instructions on the web. Procedural knowledge does not have to be stored in dedicated databases but it can seamlessly enrich instructional web pages and make them both human and ma-

[16]`https://github.com/paolo7/khjsdevelop` (accessed on 17/10/2017)

chine readable. To avoid exposing the user to HTML code one could imagine this tool automatically posting the HTML snippet on a blog or social website.

5.7 Conclusion

In this chapter I have presented a method for automatically acquiring a structured representation of semi-structured instructions that follows the PROHOW vocabulary. Semi-structured instructions are instructions where its components, such as steps and requirements, can be clearly identified by its formatting. This type of instructions has been identified as a promising target for know-how acquisition thanks to its large availability and ease to analyse.

To demonstrate the suitability of PROHOW to model real-world instructions I have parsed two instructional websites, wikiHow and Snapguide. Testing this approach on two different mainstream websites provides evidence of the generality of this approach, which is believed to be applicable also to other semi-structured instructional websites. However, I have not conducted further studies to determine the generality of this approach with more precision.

This experiment extracted of all the semi-structured instructions from these websites and resulted in a dataset of over 200,000 instructions. The quality of this dataset has been assessed as complete and correct with respect to the original sources. A similar parsing approach to the one used in this experiment can be used to acquire know-how from users on demand. To demonstrate this, I have developed a prototype of a web-form that can parse users' input into PROHOW data.

The main limitation of the knowledge extraction method I proposed is its inability to deal with unstructured resources, namely resources whose structural properties cannot be reliably interpreted semantically. Since many types of unstructured know-how resources are available on the web, alternative approaches to know-how acquisition have been developed. I have presented a review of these systems in Section 2.3. Although capturing know-how from a wider range of resources can be useful, the lack of structure in these resources makes this a significantly harder problem. This is the reason why the precision and recall of the data extracted by these alternative systems is lower than the one obtained with the approach proposed in this thesis. I see related work on know know-how acquisition as complementary to the method I proposed in this chapter. More specifically, the method I proposed can be used to generate a core dataset containing instructional knowledge which can be extracted with high accuracy.

This dataset could then be extended with more details generated by other know-how acquisition systems.

Chapter 6

Know-How Interlinking

In the previous chapter, an efficient method for extracting the predicates of the PROHOW model from semi-structured instructions was presented. These predicates provide information on how the various entities within a set of instructions relate to each other. However, they do not provide information on the relations that can exist across different sets of instructions or link to other types of data. In other words, the results of the previous chapter only describe isolated sets of instructions, and do not provide means for reasoning about them as part of a more general knowledge framework.

Instructional knowledge, however, is not isolated, and many relations exist with other types of knowledge. Relations of the types represented by PROHOW can be discovered between different sets of instructions, for example, when instruction set *A* describes how to perform a step of instruction set *B*. Relations to non-procedural knowledge can also be found; for example, an ingredient required by a set of instructions can be related to its description in a factual knowledge base.

In the context of ontologies, or computational knowledge bases, terms defined in a vocabulary carry a standard meaning by themselves, such as the term `prohow:task` defined in the PROHOW vocabulary. Databases, however, are typically populated with entities that are not defined in any vocabulary. For example, it is not possible to infer the meaning of entity `:e/01` without any other contextual information. The meaning of these entities, which cannot be derived from their identifier, is therefore defined by their relation with other entities. For example, the relation `rdf:type` between `:e/01` and `prohow:task` allows us to infer that `:e/01` represents a task. In such a system, it can be said that relations define the meaning of entities, and that adding more relations to an entity makes its semantic interpretation more informative.

6.1 Problem Description

The main challenge addressed in this chapter is how to link PROHOW instructions extracted from single web pages to additional knowledge sources. For example, the PROHOW representation of a cooking recipe might detail the ingredients used in the recipe. However, it might be unable to determine whether the recipe is vegetarian, as it does not contain information on whether any of the ingredients should be considered "meat", information which might be available in another dataset. This problem can be rephrased as the goal of making PROHOW instructions semantically more informative by integrating related datasets. In an effort to achieve this goal, I formulate the generic hypothesis that the PROHOW representation of instructions can be automatically integrated with external data both of the procedural and factual kind, with a level of quality acceptable by humans. In order to demonstrate that this level of quality has been achieved, I will compare the results of the integration system with equivalent relations already being used by humans. More specifically, I will compare them with equivalent relations found on the wikiHow website.

This generic hypothesis is split into the following two more specific hypotheses. The first hypothesis aims to demonstrate that different PROHOW representations of instructions can be integrated automatically.

Hypothesis 4. *The proposed data integration system discovers* PROHOW *relations between the* PROHOW *representations of instructions with higher precision and recall than existing equivalent relations already being used by humans.*

The second hypothesis aims to demonstrate that the PROHOW representation of instructions can be integrated with external non-procedural datasets:

Hypothesis 5. *The proposed data integration system discovers* `rdf:type` *relations between the* PROHOW *representations of instructions and DBpedia with higher precision and recall than existing equivalent relations already being used by humans.*

The integration of multiple datasets offers several advantages, most notably an increase in the richness of the data. For integrated datasets, it is typically true that the "whole is greater than the sum of its parts". All of the query answers that were computable in the original datasets are still computable after integration, as long as provenance information does not get lost and it is known which facts were contained in which dataset. But in addition to that, some answers that could not be computed by any of the original datasets might be answerable by their integration. The potential benefits of integrating the PROHOW dataset with external datasets is formulated in the following hypothesis:

Hypothesis 6. *The proposed integration between a* PROHOW *dataset and DBpedia discovers input/output relations between tasks with higher precision and recall than existing equivalent relations already being used by humans.*

An input/output relation between two tasks can be seen as the relation between a tasks that requires a certain object as an input, and another task that produces the same type of object as an output. For example, an input/output relation can be discovered between an instruction set that produces a pancake, such as "How to Make a Pancake" and an instruction set that requires a pancake as an ingredient, such as "How to Decorate a Pancake".

It should be noted that the relations mentioned in hypotheses 4, 5 and 6 do not consider the context in which specific users create or use such relations, but instead aim to capture a standardised representation of them. For example, different users might have different opinions on whether two sets of instructions should be linked. At the same time, however, websites like wikiHow try to standardise the creation of such links by providing guidelines[1] that its community should try to converge to.

Many choices need to be taken when formalising a domain into a dataset and publishing it online. While each of those choices can impact the final result, measuring its quality is not trivial. This endeavour can be divided into two parts, the first one being the formalisation of information in a certain domain into a semantic dataset, and the second being the publishing of this dataset online. Certain aspects of a formalisation can be measured both quantitatively and qualitatively, for example in terms of precision, completeness, and relevance to the domain of interest. The previous and the current chapters provide an evaluation against such measures when describing how the data has been created.

The second part of the endeavour is the act of publishing data online. Assuming that this is done to enable the reuse of this data, then this data can be seen as *open data*, which can be qualitatively measured in terms of the 5-Star Open Data publishing scheme.[2] With respect to this scheme, the following hypothesis can be formulated:

Hypothesis 7. *Data generated by the proposed knowledge extraction and knowledge interlinking method is 5-Star Open Data if published on the web under an open licence.*

[1]`https://www.wikihow.com/Weave-the-Web-of-Links-on-wikiHow` (accessed on 17/10/2017)

[2]`http://5stardata.info/en/` (accessed on 17/10/2017)

Section 6.8 describes this ranking system and the notion of open data, along with a validation of this hypothesis.

6.1.1 Data Integration as Linked Discovery

Data Integration can be seen as the generic problem of integrating multiple datasets. This term has often been used to refer to one of two specific sub-problems, which I call *schema-level* integration and *record-level* integration. The goal of schema-level integration is to be able to make different datasets interoperable by providing a unified view that allows them to be queried simultaneously. In traditional relational databases, this refers to the integration of the database schema (Lenzerini 2002). In semantic databases, this corresponds to activities such as *ontology alignment*, or *ontology matching* (Otero-Cerdeira et al. 2015).

The second meaning of the term data integration, namely record-level integration, is the one that I will use in this thesis. Record level integration is concerned with discovering relations between the specific data records that populate a database. Typically, this involves discovering when two records are equivalent, a process called *record-linkage* (Fellegi and Sunter 1969), in order to merge them and avoid duplication. This is a common process when integrating heterogeneous web-data (Michalowski et al. 2004).

In this thesis, I focus on a type of record-level integration called *link discovery*, which generalises *record linkage* to semantic link types other than equivalence. In the context of RDF data, link discovery can be seen as the process of discovering relations, or *links*, between the RDF entities in the datasets. I adopt an equivalent definition of link discovery as the one formulated by Ngonga Ngomo (2013). In this formalisation, link discovery is the problem of discovering links of a specific type α between a set of source entities S and a set of target entities T. To define this problem, we can define two sets of entities $\dot{S}$ and $\dot{T}$ denoting the domains of interest, such as "cities" and "countries". We can then define a link type α as a function $\alpha : \dot{S} \times \dot{T} \rightarrow \{0,1\}$ that maps pairs of values $< s,t >$, with $s \in \dot{S}$ and $t \in \dot{T}$, to the value 1 if a link of type α from s to t would be semantically correct, and to the value 0 otherwise. For example, if α denotes the relation of a "city s being located within country t", then $\alpha(< \text{Paris}, \text{France} >) = 1$ and $\alpha(< \text{London}, \text{France} >) = 0$. Given two subsets of the domain $S \subset \dot{S}$ and $T \subset \dot{T}$, the goal of a link discovery system is to discover the set of pairs $G \subset T \times S$ such that $< s,t > \in G \iff < s,t > \in T \times S \wedge \alpha(< s,t >) = 1$. Given our previous example of a link function α and the sets $S = \{\text{Rome}, \text{Paris}, \text{London}\}$ and $T = \{\text{France}, \text{Italy}, \text{Germany}\}$, the goal of link discovery would be to generate the set of pairs $G = \{<\text{Rome}, \text{Italy}>, <\text{Paris}, \text{France}>\}$. Having reached an abstract definition of a link discovery problem we can now analyse the two main issues than arise when attempting to solve this problem computationally, namely the problem of quality and the problem of efficiency.

6.1.1.1 The Quality Problem of Link Discovery

In a real world scenario, it is typically impossible to obtain the exact set G of links to discover. In this case, the objective of a link generation system is to generate a result set of links $\widetilde{G} \subset \dot{S} \times \dot{T}$ which approximates the correct set G as much as possible. The set G can then be called the *gold standard* that a linking system tries to approximate. The quality of this approximation is measured by combining the *precision* and *recall* measures. These measures are the ones that will be used to compare two population of links, as described in Section 3.2.2. I will now present the formal definition of precision and recall that I have adopted in the context of populations of links.

Precision The precision measure is a function $P : (\dot{S} \times \dot{T}) \times (\dot{S} \times \dot{T}) \to [0,1]$ defined as the ratio between the number of correctly discovered links and the total number of discovered links: $P(\widetilde{G}, G) = \frac{|\widetilde{G} \cap G|}{|\widetilde{G}|}$.

Recall The recall measure is a function $R : (\dot{S} \times \dot{T}) \times (\dot{S} \times \dot{T}) \to [0,1]$ defined as the ratio between the number of correctly discovered links and the total number of discoverable correct links: $R(\widetilde{G}, G) = \frac{|\widetilde{G} \cap G|}{|G|}$.

Precision and recall can be combined in a single measure of the quality of an approximate link discovery system. This single measure is the result of a function $\theta : [0,1] \times [0,1] \to [0,1]$ which combines the precision and recall measures such that a value of 1 is given to the highest precision and recall: $\theta(1,1) = 1$ and a value of 0 is given to the lowest precision and recall $\theta(0,0) = 0$. Moreover, θ should not decrease for higher values of precision and recall: $y > z \implies \theta(x,y) \geqslant \theta(x,z) \wedge \theta(y,x) \geqslant \theta(z,x)$ for any value of y, x and z in $[0,1]$. The most common θ function is the harmonic mean H, which is used by the F-measure F to combine precision and recall into an *accuracy* measure:

$$F(\widetilde{G}, G) = H(P(\widetilde{G}, G), R(\widetilde{G}, G)) = 2\frac{P(\widetilde{G}, G) * R(\widetilde{G}, G)}{P(\widetilde{G}, G) + R(\widetilde{G}, G)}$$

6.1.1.2 The Efficiency Problem of Link Discovery

The fundamental problem of generating links between resources is how to accurately discover whether two resources should be linked, and optionally how. In any realistic scenario, however, there is a second practical problem related to the computational cost of considering all possible links between two sets of resources, since the possible number of links grows as the product of the cardinality of those sets. A typical solution to this problem is to divide the source and target sets into smaller subsets, and then only consider links between certain subsets. This type of approaches alleviates memory constraints when the whole set of links cannot be loaded in memory, but individual subsets of it can (Hassan et al. 2015).

6.2 Related Work on Link Discovery

An extensive body of work already exists in discovering links between equivalent entities (Winkler 2006). Existing approaches to link discovery in the context of web data were reviewed by Nentwig et al. (2017). Typically, the discovery of equivalence relations is based on the assumption that two different records of the same entity, such as different URIs referring to the same concept, will share similar properties. For example, we might expect two records of the same person to have similar name and address attribute, and two records of the same lake to have similar geographical coordinates.

A number of systems have been developed to perform equivalence linking, such as SLINT+ by Nguyen and Ichise (2013) and KnoFuss by Nikolov et al. (2007). The Silk tool, presented by Bizer et al. (2009b), can be used to discover links other than equivalence instead. Silk can be configured to consider a number of similarity criteria, such as string similarity measures, to create links between entities. Although such links do not need to be equivalence links, they are based on similarity properties. For example, Silk can be used to link film directors in dataset *A* to the films they directed in dataset *B* by discovering an implicit equivalence between the directors described in *A* and the directors described in *B*.

The LIMES system, developed by Ngomo and Auer (2011), is designed to discover links between entities based on their similarity if this similarity can be represented in a metric space. This system has been integrated with ORCHID, presented by Ngonga Ngomo (2013), which is optimised to discover similarities between geolocated resources. The LIMES system can also be said to be primarily focused on the discovery of equivalence relations, as it is based on similarity measures.

The link discovery problems I address in this chapter, such as discovering whether one task is a step of another, are not equivalence links. Moreover, these types of links represent complex semantic relations which cannot be easily reduced to a set of similarity measures, such as string similarity. After an initial attempt at using the Silk tool I discovered that a more flexible approach was required. In the next sections, I will describe the method I developed to discover procedural links.

6.3 A method for the DBpedia Integration

Integrating human instructions with DBpedia involves finding links between procedural and declarative knowledge. Two of the most common relations at the intersection of these types of knowledge are the concepts of inputs and outputs. The task "make a pancake", for example, can be seen as a task which outputs an object of type "pancake" and requires, among others, the ingredient "milk" as an input. As discussed in Chapter 4, the PROHOW model does not explicitly represent objects. In PROHOW terms, therefore, the task "make a pancake" can be seen as one method to complete the task "obtain a pancake" and that can only be done after completion of the task "obtain milk".

With the term *DBpedia link* I will refer to a link between a task t and a DBpedia resource r, such that task t can be interpreted as the task of obtaining an object of the type described by resource r. Although t and r are semantically different, it will be assumed that their natural language descriptions will share similar textual features. Having obtained a textual representation of a task t, such as "one cup of milk", I tokenise it into a list of words and then attempt to identify a subset of it that better identifies the type of the required object, for example the word "milk". To identify the relevant subsets of the label I perform syntactic parsing to identify the noun phrases in the label. This process is used to discard other parts of a sentence which are less useful in discovering DBpedia links, such as verb phrases. Part Of Speech (POS) tagging is then used to select nouns and adjectives from each noun phrases. Overall, this process generates a number of subsets of the original label which are deemed likely to contain the textual description of DBpedia entities.

The textual label of the task t, or a subset of it, is used by the DBpedia Lookup service[3], a service that allows DBpedia entities to be searchable by keywords. In the DBpedia Lookup service, resources are matched to keywords not only if such key-

[3]`https://github.com/dbpedia/lookup` (accessed on 17/10/2017)

words appears in the title of its corresponding Wikipedia page, but also if they appear in the text of a Wikipedia anchor link to that page. The textual label of a requirement task t can be interpreted as a set of keywords to search for the most related DBpedia resources. The number of resources returned by the DBpedia Lookup service can be limited to a maximum. When a DBpedia resource is returned, my method computes a score based on a measure the textual similarity between label t and the label of the resource. The remainder of this linking method is performed in a different way depending on whether t represents an input or an output of the process.

This type of integration can be seen as a link discovery problem between the source set S of input and output tasks and the target set T of DBpedia resources. With the terms *DBpedia input links* and *DBpedia output links* I will refer, respectively, to DBpedia links from the input and output tasks of the set of instructions. The type of link α that will be considered is the relation between a task that involves obtaining or creating a certain object, and a DBpedia resource representing a type of that object. Since the set of such tasks S in the HOWDB dataset has cardinality $5 * 10^5$, and the set of DBpedia[4] resources T has cardinality $6 * 10^6$, the space of all possible DBpedia links $|S| * |T|$, which has cardinality $3 * 10^{12}$, can be considered too large to be efficiently explored in its entirety. The use of the DBedia Lookup Service is an approach to solve the efficiency problem of link discovery. It is not necessary to consider links to every possible DBpedia resource, but only to those returned by the DBpedia Lookup service. If this service, on average, returns n resource candidates, with $n \ll |T|$, only $n|S|$ links would need to be considered.

6.3.1 Input-Output Matching

The union of input and output links to DBpedia effectively results in new links between tasks which will be called I/O links. An I/O link exists between a task a, such as "prepare pancake batter" and another task b, such as "pancake batter", requirement of a task c, such as "make a pancake", if task a outputs an object of a type that is subsumed by the type of the object required by b. This match between inputs and outputs is a common approach used in service composition (Constantinescu et al. 2004). This approach is based on the principle that the output of a service can be used as the input of another service, if the type of the output is subsumed by the type of the input. It is assumed, therefore, that the type of the input refers to the most generic class of objects

[4]Official figures of the DBpedia version 2016-04 taken from `http://wiki.dbpedia.org/dbpedia-version-2016-04` (accessed on 17/10/2017)

that the input is compatible with.

When analysing a set of instructions, I assume that the description of the requirement of a set of instructions defines the type of its input. More specifically, if the label of a task t involves obtaining an object described as being of type x, then obtaining any object of a type that is subsumed by type x can be seen as accomplishing task t. For example, the requirement “pancake batter” would correspond to an input of type “pancake batter”. This input can be matched with outputs of more specific types, such as “vegan pancake batter”, as they are subsumed by the type of the input. However more general types, such as “food”, cannot be matched.

6.3.2 Input Integration

If the analysed task t, such as “one cup of milk”, is a requirement of a main set of instructions, such as “how to prepare a pancake”, then t will be considered as defining an input of the process. The labels used in wikiHow and Snapguide to refer to these requirements directly describe the object that needs to be obtained and typically do not contain verbs. A link with a DBpedia concept is created if the DBpedia Lookup service discovers an entity with a sufficiently similar label to t. To determine this similarity I have empirically chosen a threshold of 0.8 for the Levenshtein distance between the label of task t and the label of a DBpedia concept. If no such DBpedia concept is found, a number of steps are taken to remove unnecessary words from the label of task t, and then the search is computed again for subsets of the label of t. The steps followed are the following:

1. The English stopwords “the”, “an” and “a” are removed.
2. Parentheses, and their contents are removed.
3. Units of measure are removed. A unit of measure can be detected when a unit of measure word, such as “cup” or “liter” is followed by the particle “of”. The list of units of measures used is given in Appendix E.1.
4. Symbols and digits are removed.
5. Adjectives which do not modify the type of an object, such as “grated”, “diced” or “melted” are removed. This is an optional step to inject domain knowledge, in this case in the cooking domain, that can help improving the performance of the integration. The list of adjectives used is given in Appendix E.3.

6. Connectives "and" and "or" and the symbols "&" or "/" are used to split the label into multiple sub-components, which are then analysed separately.

These steps are followed incrementally until an entity with a sufficiently high similarity score is found, or until all of these steps have been tried. If none of the above mentioned transformations leads to the discovery of a DBpedia resource with a sufficient level of textual similarity, then task *t* will not be linked to any DBpedia resource. If multiple DBpedia resources with sufficient textual similarity are found, the one with the highest score is selected.

If a requirement task is decomposed into one or more components, the type of the connective defines whether all of the components are needed or if only one of them is sufficient. Disjunctive connectives, such as "or", result in a `prohow:has_method` relation between the task and its components. For example, the sub-components *a*: "eggs" and *b*: "milk" can be considered methods of a task *x*: "eggs or milk", as *x* can be completed, for example, by only accomplishing task *a*. Similarly, conjunctive connectives, such as "and", result in a `prohow:has_step` relation between the task and its components. A task *x*: "eggs and milk" can only be considered complete, in fact, if both of its components *a*: "eggs" and *b*: "milk" are complete.

6.3.3 Output Integration

While the inputs of a process are typically listed as the requirements of its set of instructions, the outputs are not explicitly defined. The outputs, and in general the effects, of a process can be many. For example, the process of making a pancake might result not only in the creation of a pancake, but arguably also in a greased pan and in a release of heat. The analysis of the outputs described in this thesis does not attempt to identify all the possible outputs of a task. Instead, it focuses on what can be called the *main output* of a task. The main output of a set of instructions, if present, is the object that the task is primarily intended to generate. It will be assumed that the main output of a task is the object mentioned in the description of a task. For example, the main output of a task on how to "make a pancake" will be considered a "pancake" and not "heat" because the word "pancake" appears in its description while "heat" does not.

The definition of what a main output is suggests that its textual representation can be found within the label of the task. While many words can appear in the label of a task, not all of them represent outputs. A task will be considered as potentially having a main output if the verb found in its label is a *creation verb*. A creation verb is verb which semantically implies the creation of an object, such as the verbs “create”, “produce” and “build”. The list of the creation verbs considered in this experiment is given in Appendix E.2. The noun phrase that acts as the syntactic object of the creation verb is considered as a candidate output. For example, in the task label “make a pancake” the word “pancake” would be considered as a possible output because the verb “make” is a creation verb. An attempt is then performed to link the text extracted from this noun phrase to a DBpedia resource using the same approach previously described for the input integration.

6.3.4 A Note on the Semantics of the Links to DBpedia

When linking PROHOW tasks and DBpedia entities, different approaches are possible. I will now discuss the semantic differences of such approaches, along with their advantages and disadvantages. It should be noted, however, that these different approaches can be considered equivalent with respect to this thesis, and therefore they will not be discussed any further beyond this section.

If the main goal is to have a simple and compact representation, a task such as “1 liter of milk” can be related to DBpedia entities, such as `dbr:Milk`,[5] with the `rdf:type` relation, which relates instances with their class. Although practical in some contexts, this formalisation can be said to be semantically imprecise. In fact, the task of obtaining a requirement is not the same as the requirement itself. For example, an object of type “milk” can have a weight, but it cannot be said to have been accomplished. Conversely, the “act of obtaining milk” does not have a weight, but it can be said to have been accomplished.

To represent these links more correctly, the object that is being obtained should be separated from the task that involves obtaining it. Given that PROHOW does not represent objects, an extension of it would be required. This could be done for example by adding a new relation such as `prohow:obtains_object`, which links the tasks which represent obtaining an object, with the object itself. Using this model, the object obtained by a task can now be described with additional properties such as a type or a quantity. The RDF data in Listing 6.1, for example, describes the task “1 liter of milk” (`:requirement_milk`) that involves obtaining an object (`:milk_obj`) defined as something which is “milk” (`dbr:Milk`) and that amounts to the quantity of 1 litre, using the Quantities, Units, Dimensions and Types (QUDT) schema.[6]

[5] I use the `dbr` namespace defined in Table 2.1 to refer to DBpedia resources.
[6] `http://www.qudt.org/` (accessed on 17/10/2017)

Listing 6.1: The PROHOW representation of the requirements of the pancake recipe.

```
:requirement_milk rdfs:label "1 liter of milk" .
:requirement_milk prohow:obtains_object :milk_obj .
:milk_obj rdf:type dbr:Milk .
:milk_obj qudt:hasQuantity :milk_quantity .
:milk_quantity qudt:quantityValue :milk_quantity_value .
:milk_quantity_value qudt:unit unit:L .
:milk_quantity_value qudt:numericValue "1" .
```

The topic of a Wikipedia page, such as the page for the concept of "milk",[7] is represented in DBpedia as a "resource", in this case: `dbr:Milk`. It is important to state that DBpedia resources were not created to be used as classes. Arguably, while the DBpedia resource for milk could be interpreted as the class of things that are milk, the DBpedia resource for the city of Edinburgh, `dbr:Edinburgh`, should not be interpreted as the set of things which are Edinburgh. The reason behind this semantic difference is that the Wikipedia page for "milk" describes the abstract concept of milk, while the Wikipedia page for "Edinburgh" describes the individual and tangible entity of this specific city. Therefore, although direct `rdf:type` links to DBpedia resources can be practical in certain domains, they are not guaranteed to be correct. DBpedia itself treats certain resources similarly to classes, linking instances to their class with the `dbo:class` relation. However, the choice of using a proprietary relation instead of the most common `rdf:type` relation is a clear indication that these resources cannot be assumed to be classes, at least in the RDFS definition of this term.

An attempt to solve this problem comes from the Product Types Ontology (PTO),[8] which provides a set of alternative URIs to DBpedia resources to be used as classes. For example, the URI `pto:Milk` can be used instead of `dbr:Milk` to denote the class of things that are milk. Unfortunately, PTO does not currently discriminate between DBpedia resources denoting concepts and those denoting individual entities. In fact, the city of Edinburgh is also considered a PTO class (`pto:Edinburgh`), and more specifically a "product or service". As a result of this, the core semantic problem of whether a DBpedia resource can be considered as a class or not remains unsolved.

[7]`http://en.wikipedia.org/wiki/Milk` (accessed on 17/10/2017)
[8]`http://www.productontology.org/` (accessed on 17/10/2017)

6.4 A method for Decomposition Links

As the popularity of step-by-step instructions show, most complex tasks can be decomposed into simpler steps so that an agent unable to complete a task as a whole might be able to complete it by completing each of its steps individually. However, it is not uncommon that the steps themselves involve complex operations that the agent is still unable to accomplish. For example, a set of instructions on how to "apply for a job" might include the step "prepare a resume". An agent that is not capable of preparing a resume might wish for this step to be further decomposed into even simpler sub-steps, such as "list your previous work experiences". It is apparent therefore, that the decomposition of tasks into sub-tasks does not result into a two-layer structure made of either complex or simple tasks. Instead, task decomposition can be seen as a multi-layer task graph where tasks at each level can be decomposed into progressively simpler tasks or composed together into progressively more complex ones.

A task network describing a significant portion of human know-how would contain too much information to be displayed to a human as a single document. It is necessary, therefore, to divide this network into smaller subsets of a more convenient size. These subsets are what we call sets of instructions, and each of them describes a limited number of sub-tasks of a specific task. Although many related tasks cannot be described in a single set of instructions, links to them need to exist if we want to be able to recreate and traverse the original task network. For example, details on how to prepare a resume do not have to be included in the set of instructions on how to apply for a job, as long as they can be discovered by following a link between the two sets of instructions.

In a single set s of instructions we can identify a *top level* task and a set of *bottom level* tasks. The top level task t of s is the main task that the instructions are designed to describe. Within s, the main task is decomposed into sub-tasks, but it is not used to decompose any task itself. Conversely, the bottom level tasks represent the locally most fine grained level of task decomposition. Within s, no bottom level task is decomposed into simpler tasks, while they are all used to decompose more complex tasks. With respect to the PROHOW statements contained in a single set of instructions, Equations 6.1 and 6.2 define, respectively, top and bottom level tasks.

$$\begin{aligned} &\text{top}(x) \iff (\exists y.\text{has\_step}(x,y) \vee \text{has\_method}(x,y)) \wedge \\ &(\neg\exists y.\text{has\_step}(y,x) \vee \text{has\_method}(y,x)) \end{aligned} \tag{6.1}$$

$$\begin{aligned} &\text{bottom}(x) \iff (\exists y.\text{has\_step}(y,x) \vee \text{has\_method}(y,x)) \wedge \\ &(\neg\exists y.\text{has\_step}(x,y) \vee \text{has\_method}(x,y)) \end{aligned} \tag{6.2}$$

Tasks which are not of either of these two types will not be considered in this phase, as they are already linked with both more complex and less complex tasks, or they are requirement tasks that are already considered in the DBpedia integration. The goal of this phase, instead, is to link the top level tasks and the bottom level tasks between different sets of instructions. More precisely, we can define this phase as a link discovery problem between the set S of bottom level tasks and the set T of top level tasks for a relation α. Link discovery relation $\alpha(< s,t >)$ equals 1 if the target element t could be considered either a step or a method of the source element s according to the PROHOW definition of steps and methods defined in Section 4.7. Links of this type will be referred to as "decomposition links"

A top level task t describing "how to prepare a resume", for example, could be considered a possible method of the bottom level task a "prepare a resume", as the completion of task t would be sufficient to complete task a. However, task t could be considered only a step of a task b "prepare a resume and attach it to the application form", as the completion of task t would complete part of task b, but not all of it. Given the difficulty in establishing with high accuracy whether the execution of a top level task is sufficient or not to complete a bottom level task, the safest assumption is for the top level task to be considered a step of the bottom level one. In the PROHOW model, a decomposition link between a top level task t and a bottom level task s can be represented as t being the step of one possible method m for completing s, along with potentially other steps. Listing 6.2 represents a possible representation of this link. Other potential tasks that need to be performed are represented with the blank node `_:others`. Task t requires the task `_:others` as other steps might need to be executed before t. A blank node is an RDF feature which can be used to express the existence of an entity in an RDF graph without having to explicitly refer to its URI. Without an explicit mention of these other potential tasks, step t could be considered sufficient to complete method m.

Listing 6.2: The PROHOW representation of a decomposition link between top level entity :t and bottom level entity :s.

```
:s prohow:has_method :m .
:m prohow:has_step :t .
:m prohow:has_step _:others .
:t prohow:requires _:others .
```

Following the previous definition of a link discovery problem, two sub-problems need to be addressed: obtaining links with high quality and in an efficient way. The problem of discovering links with high quality can be posed as the problem of discovering whether two tasks should be linked. More specifically, given a top level task t and a bottom level task s, along with a set of features describing each task, such as their label, a classifier should determine whether the two tasks should be linked or not. This problem does not have a simple and precise solution, as the decision on whether two tasks should be linked is strongly dependent on the semantics of the tasks. In this situation, a machine learning classifier can be trained to make this decision based on a set of given features.

In this classification problem, each task can be defined with the following three characteristics:

- a label;
- the "context", or more specifically the labels of the other entities in the same set of instructions;
- the category of the set of instructions the task belongs to.

Moreover, the Inverse Document Frequency (IDF) (Robertson (2004)) of each word can be used to determine how rarely a word appears across different sets of instructions. The occurrence of the same word across two labels can be considered more significant for higher IDF values of the word.

Typically, the labels of a top level task are short and concise, while the labels of the bottom level tasks can extend to several sentences. Therefore, it is often the case that the label of the top level task is a sub-string of the label of the bottom level task, and only in rare occasions would the two labels be of comparable length. When comparing two tasks, their characteristics can be combined into a number of features. A few examples of features that can be used are the following:

- the ratio of words of the label of t found in the label of s;
- the average IDF scores of the words in common between the labels;
- the number of common words in the tasks contexts;
- the average IDF scores of the words in common between the contexts;
- the number of categories and super-categories in common;

- whether the sentence found in the label of t and its closest match in the label of s are both under or outside the scope of negation.

This last feature can be useful, for example, to prevent a link from task "don't make a pancake" to the task "how to make a pancake".

In order to create a training dataset, pairs of tasks need to be randomly selected and manually annotated with information on whether these pairs should be linked or not. This set of manually annotated task pairs, along with their related features, can then be used to train a classifier to predict whether two task should be linked or not. However, careful consideration should be paid when deciding from which distribution to draw candidate task pairs due to the problem of class imbalance, which will be discussed next.

6.4.1 The Problem of Class Imbalance

In the HOWDB dataset, if we randomly select pairs of top and bottom level tasks, we will quickly realise that virtually all of them are pairs which should not be linked together. As a result of this, it would be prohibitively expensive to manually annotate a set of randomly selected links which contains a number of linkable pairs of tasks deemed sufficient for classification purposes. The reason of this problem is class imbalance.

Class imbalance occurs when entities of a certain class are significantly underrepresented in a dataset, and it is a well known problem in the Machine Learning community (Japkowicz and Stephen (2002)). For example, class imbalance is common in fraudulent economical transactions, which arguably occur only in a minority of cases, but that can be very costly if undetected. While class imbalance can affect multiclass classification problems, it will be discussed here from the point of view of binary classification. In a dataset containing datapoints that are either of class 1 (positive), or class 0 (negative), assuming that 1 is the minority class, we can measure class imbalance as the ratio of positive datapoints in the whole dataset (Liu et al. (2009)). For example, a dataset containing 1 positive datapoint for every 10 datapoints has an imbalance ratio of $1 : 10$. While an imbalance ratio of $1 : 2$ denotes perfect balance in a binary classification scenario, the problem of class imbalance increases as this ratio becomes smaller. The most common danger posed by class imbalance is the tendency of many ML algorithms to classify all datapoints as the majority class; strategy that would result in a classification with high accuracy. Approaches to deal with class imbalance often attempt to counter this tendency, for example by increasing the penalty generated by false negatives of the minority class. It is also possible to use training data where both classes are equally represented, as long as higher weight is given to the misclassification of data points of the majority class. A review of these types of approaches is given by Japkowicz and Stephen (2002).

Adding different weights to different classes can help keeping the training data size requirements within practical limits, but it does not solve all of the problems. In order to learn how to detect the minority class efficiently, a learning algorithm will need to discriminate between borderline data points with high precision. In this thesis, borderline data points belong to a borderline class, defined as a class that is (1) statistically more similar to the minority class than the majority class, (2) contains a significant number of data points of the majority class and (3) its size is similar to the size of the minority class but significantly smaller than the size of the majority class.

Going back to our linking scenario, the space of possible links is typically very large compared to the specific kinds of links one is interested in. The set of *linkable* pairs of entities can therefore be said to be the minority class. In this scenario, one example of borderline class is the class of pairs of tasks that contain all the words of the target entity in the source entity. Intuitively, this criteria is not sufficient to discriminate a positive link from a negative link, as the pair [“if you do not have a job, apply for a discount”,“apply for a job”] should not be linked, although all the words in the label of the target task appear in the label of the source task. Since most task pairs do not have many words in common, we can expect this borderline class to be under-represented within the majority class. On the other hand, this property is very common in the minority class, and most of its data points would also belong to the borderline class. A learning algorithm that utilises a randomly sampled set of data points form the majority class would then face serious difficulties in detecting borderline cases, due their relative scarcity in the negative set. Although this scarcity can be overcome by using a larger training set, the extreme level of class imbalance would make this approach prohibitively costly. If ignored, this problem can lead to a significant loss of accuracy as the classifier might learn how to classify the borderline class instead of the minority class. This can happen if the borderline class is easier to detect than the minority class, and if not enough data points are present which belong to both the majority and the borderline classes.

A possible solution to this problem is applying a hierarchical classification system where the highest level of classification does not filter out borderline cases, filtering that happens instead in subsequent classifiers. The links selected by the first classifier have a very different distribution from the original data, as the positive and borderline examples are now much more numerous. This set of links can then be used to create a second training set, which attempts to separate the borderline cases from the minority class. This approach can be repeated multiple times, resulting in multiple hierarchies of classification, to obtain a progressively more precise classifier. The main drawback of such an approach, however, is the necessity to generate a new training set for each level of classification.

A variant of this approach is possible by substituting the top level of the classification hierarchy with a heuristic classifier (Clancey (1985)) if a suitable high accuracy heuristic is known. Unlike a Machine Learning classifier, a heuristic classifier does not involve learning and therefore does not require a training dataset. Instead, it uses a pre-defined algorithm or set of rules to classify data points. In the next section I will discuss my method for link discovery which adopts this variant of hierarchical classification.

6.4.2 Hierarchical Classification

The proposed method for discovering decomposition links adopts a hierarchical approach composed of two filters: a crude and a fine-grained one. The crude and fine-grained filters are based, respectively, on a heuristic classifier and on a Machine Learning classifier. Figure 6.1 illustrates how these filters are combined together to solve the link discovery problem. The main goal of the first filter, the crude one, is to significantly reduce the space of all possible links into a subset of more manageable size. A secondary goal of this filter is to address the problem of borderline classes, by generating a subset of potential links where a higher ratio of borderline classes is represented.

The proposed type of crude filter is based on efficient text indexing and search systems. In the first phase, all the labels of the top level and bottom level tasks are indexed in a text search system. In the second phase, the label of each top level entity t is used as a search query in a text search system to retrieve the set B_t of bottom level entities which have the highest label similarity with t. When considering a top level task with label “how to prepare a resume”, for example, the system might search for bottom level tasks with a label containing the words “prepare” and “resume”.

A threshold criterion should limit the average size of the sets B_t across all the top level tasks T to a value n significantly smaller than the set S of bottom level entities

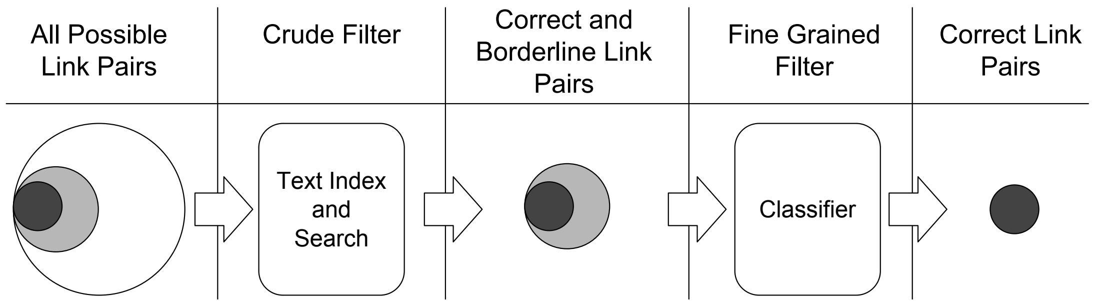


Figure 6.1: Diagram of the decomposition links discovery processes. The sets of correct, borderline-incorrect and incorrect link pairs is represented, respectively, with the colors black, grey and white.

$n << |S|$. This can be done, for example, with a hard limit on the cardinality of B_t, or by only including in this set elements with a textual similarity above a certain value. A good threshold criteria should balance the need to reduce the size of B_t while keeping the number of correct links for t not included in B_t to a minimum.

The set F of all the candidate links between top level entities t and their similar entities in B_t can then be processed further by the fine grained filter. While the original set of all possible links had cardinality $|T| * |S|$, the candidate subset F has the smaller cardinality $n|T|$. Moreover, F has different statistical properties than the original set of all possible links. In particular, links in F have a higher textual similarity. This property will reduce the risk that a classifier would learn to give too much importance to text similarity alone, effectively learning how to classify borderline links instead of the correct ones.

Having obtained the filtered set of candidate links F, it is now possible to manually annotate a small subset of randomly selected links from F with information on whether each link should be accepted as correct or not. These annotated links, along with their computed features, can be used as a training dataset to train a classifier to detect the correct links in F. Once trained, this classifier is used to filter the final set of correct decomposition links from F. The manual annotations discussed in this chapter have been gathered through the work of trained annotators. However, these annotations could have also been gathered by crowdsourcing. In particular, crowdsourcing could be used to detect whether particular sub-populations of users have different opinions on the correctness of the above mentioned links.

6.5 Implementation and Experiments

A concrete implementation of this method was developed as an integrated set of Java applications[9] along with the DBpedia Lookup Service, the Virtuoso[10] triplestore and the text search engine Apache Lucene.[11] The HOWDB dataset was loaded in the Virtuoso triplestore, and from there, the textual label of each top and bottom level tasks, their category, and information on whether they are top or bottom level tasks was extracted and loaded into a Lucene index.

DBpedia links were generated using the proposed method. The number of matches returned by the DBpedia Lookup Service were limited to a maximum of 60. This threshold was empirically chosen after observing that correct matches, if present, tend to appear very close to the top of the result set. The analysis of requirements made use of a list of standard labels for units of measure, such as “spoons”, “ml” and “grams”, which can be easily obtained on the web. Six common adjectives such as “sliced” and “grated” were also removed from the labels, as they typically do not modify the type of the ingredient they refer to.

The decomposition links were generated as follows. For each top level task, a query obtained the set of candidate bottom level tasks containing at least one word from the label of the top level task, excluding stopwords. Each pair of tasks derived this way was added to the set of candidate links F. A vector representation of each pair of links in F, consisting of 34 features, was automatically computed. From this set, 1000 links were randomly selected and they were annotated as either correct or wrong by a human annotator. This training data was used to train a Random Forest classifier using the implementation provided by the WEKA Machine Learning library (Hall et al. (2009)). The classifier was then applied to all of the links in F to generate the final set of decomposition links.

6.6 Evaluation

The results of the DBpedia link discovery experiment can be found in Table 6.1. The precision was manually evaluated separately for the inputs (P_I) and the outputs (P_O) on 300 randomly selected links each. This evaluation was performed by a University student. This student was instructed to classify a link as wrong (1) if it linked an entity

[9]https://github.com/paolo7/Know-How-Extractor-Linker/ (accessed on 17/10/2017)
[10]https://virtuoso.openlinksw.com/ (accessed on 17/10/2017)
[11]http://lucene.apache.org/core/ (accessed on 17/10/2017)

	Inputs	Outputs	Total
Number of linked entities	255,101	4,467	259,568
Estimated number of correct links	244,896	4,391	249,185
Number of different DBpedia types linked	8,453	3,439	10,166
Precision	96% (P_I)	98.3% (P_O)	96% (P_{I+O})

Table 6.1: Results of the DBpedia integration experiment

which was not an input or an output of the process or (2) if the type of the input or output did not correspond to the linked DBpedia type. The recall measure was not computed, as the total number of possible DBpedia links is not known. For example, certain tasks might refer to specific objects not present in DBpedia, while others might mention multiple objects, and therefore generate multiple DBpedia links. Estimating the recall of the output links is even harder, as it is not known how many of the $215,959$ top level tasks (Table 5.1) involve the creation of an object. However, if we assume that all of the $530,352$ requirements in the HOWDB dataset (Table 5.1) should be associated with exactly one DBpedia resource, then the correct discovery of $255,101$ DBpedia input links, 96% of which correct, results in a recall of about 46.1% of the input links.

The precision of the decomposition links was computed by the classifier using 10-fold cross validation. The performance of this classifier, listed in Table 6.2, is a precision of 82.4%. In total, this classifier led to the discovery of $193,701$ links.

6.6.1 Comparison with a Human Benchmark

To understand the significance of the results of our integration experiment, we need to compare them with a relevant benchmark. To the best of the author's knowledge, the proposed link discovery system is the first to automate the linking of step-by-step instructions. Therefore, a direct comparison with a previous equivalent system is not possible. It is possible, however, to compare it with an existing manual integration effort that is being performed by the wikiHow community. The members of this community, in fact, are not only active in the creation and the refinement of individual sets of instructions, but are also actively creating links between different processes. This effort can be seen as evidence of the human benefits coming from integrating know-how. User participation in this linking effort is actively encouraged by wikiHow,[12] which does not seem to employ automatic linking systems, an assumption that was

[12]`http://www.wikihow.com/Weave-the-Web-of-Links-on-wikiHow` (accessed on 17/10/2017)

confirmed by asking members of the wikiHow community on the relevant discussion pages. These community-generated links can be analysed to obtain evidence of the level of quality that can be deemed sufficient for human use. This analysis will attach concrete figures to the human quality criteria mentioned in the hypotheses defined in Section 6.1.

To evaluate the quality of the links generated by the wikiHow community, which will be referred from now on as **WH-C** links, the first step was to identify such links in the original sets of instructions on wikiHow. This extraction was performed by collecting the linked URLs from the HTML anchor tags within the labels of the entities of each set of instructions, and by recording those pointing to other wikiHow articles. According to wikiHow's policy on external links,[13] links to sets of instructions on external websites are not generally allowed. It is possible therefore to evaluate these links within wikiHow articles alone. Links between wikiHow articles can be split into two groups. The first group consists of the links found in the requirements of a task. These links typically connect a required object to a task that can produce such an object. As such, they have the same functional role as the I/O links found by the DBpedia integration. The second group consists of the links found in the steps of a task. These links connect a step with another task which provides additional information on how to complete that step. As such, they have the same functional role as the decomposition links.

Having identified a comparable set of links, three quality metrics for each link type were computed. These quality metrics are the following:

Precision of the links Correct links should provide relevant information on how to accomplish a certain task. For example, the proposed system correctly linked the step "Avoid smoking cigarettes, cigars or pipes around the baby" with the relevant process "How to Avoid Smoking". The wikiHow community integration, instead, linked this step with the irrelevant processes "How to Smoke a Cigar" and "How to Prevent Frozen Water Pipes".

Number of links found The recall of a linking system can be said to be higher if it generates a larger number of links.

Coverage of the links This metric evaluates the number of sets of instructions which contain at least one link to another set of instructions. The fact that links are

[13]`https://www.wikihow.com/wikiHow:External-Links` (accessed on 17/10/2017)

more evenly spread across different sets of instructions can be considered a sign of a better integration.

The result of this comparison can be seen in Table 6.2. For each of the two types of links generated by the wikiHow community (**WH-C**), the precision was evaluated manually by the same student that participated in the evaluation described in Section 6.6. I divide these links into two populations **WH-C**^{D} and **WH-C**$^{I/O}$ which contain, respectively, all the decomposition links and the the I/O links in dataset of human generated links **WH-C**. For each of these populations, the precision measure was computed by manually evaluating 200 randomly selected links as either correct or incorrect, according to the respective interpretation of the links.

I have compared these two populations with populations **WH+S**^{D} and **WH+S**$^{I/O}$, which contain, respectively, the decomposition links and the I/O links generated by my integration system. The precision of population **WH+S**$^{I/O}$ was computed as the probability that both the input and the output links involved are correct: $P_{I/O} = P_I * P_O$. As such, it can be derived from the precision values shown in Table 6.1. Both P_O and P_I were manually evaluated over a set of 300 randomly selected links. The precision of the population **WH+S**^{D}, which was generated using a classifier, was evaluated using 10-fold cross validation over a manually evaluated dataset of 1000 randomly selected links.

I have computed the statistical significance of these measurements using the chi-squared test, and the differences in precision between the pair of populations **WH-C**^{D} and **WH+S**^{D}, and between the pair **WH-C**$^{I/O}$ and **WH+S**$^{I/O}$, are statistically significant. The probability of the null hypothesis, namely that the two pairs of populations have the same precision, is less than the threshold of 0.05. This difference shows a higher precision of the links generated by my system compared to the ones created by the wikiHow community.

It should be noted that the links generated by the wikiHow community only interlink wikiHow instructions. The proposed system, instead, integrates instructions both from the wikiHow and the Snapguide repositories. To make a fair comparison of the number of correct links, Table 6.2 also shows the measurements of the links generated by our system that only connect wikiHow resources (**WH**). The estimated number of correct links discovered by my system on the wikiHow dataset is of 88.5K I/O links (**WH**$^{I/O}$) and 104.6K decomposition links (**WH**^{D}). The estimated number of correct links created by the wikiHow community is of 2.9K I/O links (**WH-C**$^{I/O}$) and 72K decomposition links (**WH-C**^{D}). Following the methodology discussed in Section

	WH-C	**WH**	**WH+S**
Precision of I/O links	65%	94.3%	94.1%
Number of I/O links	4,560	93,883	183,094
Estimated correct I/O links	∼2,964	∼88,531	∼172,291
Coverage of I/O links	3,342 (1.9%)	35,169 (21%)	58,029 (27.4%)
Precision of decomposition links	71%	82.1%	82.4%
Number of decomposition links	101,496	127,468	193,701
Estimated correct decomposition links	∼72,062	∼104,651	∼159,609
Coverage of decomposition links	45,250 (27.1%)	69,859 (41.8%)	90,217 (42.6%)
Total precision of the links	70.7%	87.3%	88.1%
Total number of links	106,056	221,351	376,795
Estimated number of correct links	∼74,981	∼193,239	∼331,956
Total coverage of the links	45,999 (27.5%)	84,350 (50.4%)	114,166 (53.9%)

Table 6.2: Comparison of the wikiHow community integration (**WH-C**) with the results of our integration of wikiHow (**WH**) and of both wikiHow and Snapguide (**WH+S**)

3.2.2, the higher number of correct links in populations $\mathbf{WH}^{I/O}$ and $\mathbf{WH}^{D}$ compared to, respectively, populations $\mathbf{WH\text{-}C}^{I/O}$ and $\mathbf{WH\text{-}C}^{D}$, can be seen as evidence of higher levels of recall in the populations generated by my system.

6.7 Discussion of the Results

The result of this evaluation shows how the proposed method for link discovery generates both PROHOW decomposition relations and input/output relations within a PROHOW instructional dataset with higher precision than the existing human benchmark. An exact measure of recall could not be computed due to the lack of a practical approach to determine the total number of correct links that should be discovered. However, it is possible to perform a relative measurement of recall between multiple populations under the assumption that the total number of correct links is equal across the populations. Given that this assumption is true in my experiments, and that I discovered a higher number of correct links than the ones found in the human benchmark, I can conclude that the proposed system generated links with higher recall (as discussed in Section 3.2.2).

Given that the set of all the possible correct links is not known, it is also not possible to carefully determine if there is any particular category of links that my system systematically fails to detect. However, we can consider that my system strongly relies on string similarity. Therefore, one could expect lower levels of recall across populations of links that connect entities that are semantically related, but that are described using very different natural language expressions.

Having shown higher precision and recall for both types of links, the experiments can be said to validate Hypotheses 4 and 6. On the other hand, Hypothesis 5 cannot be validated directly, as an existing benchmark of direct links to DBpedia (or equivalently Wikipedia) resources was not found. It can be observed, however, that the precision of the discovered DBpedia links is higher than the precision of the other types of human generated links. Moreover, the quality of the I/O links, evaluated as being higher than the human benchmark, is a direct result of the quality of the DBpedia links. Therefore, these observations can be taken as evidence suggesting the validity of Hypothesis 5.

6.8 5-Star Open Data

After interlinking a PROHOW dataset with other datasets, it is possible to publish it on the web to allow other people and organisations to use it or to link to it. Publishing this type of data not only brings benefits to those that want to use it, but also to those that publish the dataset. The value of a dataset, in fact, can be considered to increase if it is used by external applications, or if it is linked to from other datasets. The DBpedia links described in this chapter not only result in the discovery of I/O relations between tasks, but they also enrich the DBpedia dataset, making it possible to discover, for example, of which resources are often used together in instructions.

The term *open data* refers to data in an open form, which can be both obtained from the web and used freely.[14] This term is associated with a movement that promotes the practice of publishing open data, with the objective of unlocking data's full potential by making its use easier and less restricted. This movement counters the opposite trend of closing or restricting access to datasets, often implemented by private companies to maintain the economic or competitive advantage of having exclusive access to the data.

Hypothesis 7 can be validated by showing that HOWDB, the dataset generated by the proposed knowledge extraction and interlinking method, would qualify for all of the five stars of the 5-Star Open Data ranking system if it was published on the web under an open licence. The HOWDB dataset is, in fact, published online[15], but under

[14] `http://opendefinition.org/od/2.1/en/` (accessed on 17/10/2017)
[15] `https://w3id.org/knowhow/dataset` (accessed on 17/10/2017)

a licence[16] which allows anyone to copy, use, modify and redistribute the data for non-commercial purposes only. Having to restrict its use only for non-commercial purposes is a result of the terms of use of the wikiHow and Snapguide websites, which do not allow their data to be used for commercial purposes. Therefore, the inability of publishing the HOWDB dataset under an open licence is not considered a flaw of the method, but only a result of a particular choice of the original data sources.

Following the methodology outlined in Section 3.2.3, I validate Hypothesis 7 by demonstrating compliance of each star of the 5-Star Open Data ranking system. The reasons why each star should be awarded are the following:

- The first star is awarded for publishing data on the Web under an open licence. This criteria is met as it is part of the assumptions behind the formulated hypothesis.
- The second star is awarded for making the data available as structured data. This criteria is met by having the data represented in the structured RDF data model.
- The third star is awarded for making the data available in a non-proprietary open format. This criteria is met by publishing the data in the Turtle format (Carothers and Prud'hommeaux 2014), which is open and non-proprietary.
- The fourth star is awarded by using URIs to denote concepts. This criteria is met because, with the exception of the literal values such as dates and strings of text, every entity and relation in the dataset is represented with a URI.
- The fifth star is awarded by linking the data to other datasets and thus provide context. This criterion, which is expressed in vague terms, needs to be formulated into a more precise one before it can be evaluated. In Section 3.2.3 I have argued that a sufficient criterion for the fifth star to be awarded to an instructional dataset is the presence of a number of links to DBpedia concepts greater than the number of sets of instructions. Having discovered $249,185$ links to DBpedia (Table 6.1), across $211,128$ sets of instructions (Table 5.1), I consider that there exists sufficient evidence for the fifth star to be awarded.

Having validated Hypothesis 7, it can be said that the data integration system described in this chapter complements the data extraction method described in the previous chapter in providing a complete method to generate 5-star Open Data resources

[16]`https://creativecommons.org/licenses/by-nc-sa/4.0/` (accessed on 17/10/2017)

from instructional web-pages. The only parts of this process that have not been described in detail are the ones pertaining the first star of the ranking system, namely how to publish data online under an open licence. This step is not described as it can be considered trivial and implementation-specific. A large number of data hosting solutions exist, and pre-made open licences, such as those distributed by the Open Data Commons,[17] are also readily available.

6.9 Conclusion

In this chapter I have presented a prototype of a link discovery system for instructional data, along with insights on how it can be built and the challenges that need to be addressed to do so. I have evaluated this method in a real-world scenario and demonstrated that it is possible to generate procedural links that significantly outperform manual community-based integration efforts. The proposed method should not be interpreted, however, as an optimal strategy for procedural link discovery, and many improvements can be made to generate a faster and more precise system.

For example, this method was further analysed by Chernov et al. (2016) in an independent stream of work. Chernov et al. validated the hypothesis that when linking instructional text, imperative verb phrases (or *actionable phrases*) are highly relevant. The approach that considered actionable phrases discovered 60% more links than the one that considered whole step sentences, and with higher precision. A difference from the experiments presented in this chapter, however, is that their evaluation was limited to the *Computer and Electronics* domain of the wikiHow website, which was analysed with a domain-specific algorithm that exploited knowledge of the terminology of that domain.

The work presented in this chapter complements the work presented in Chapters 4 and 5 in providing a complete method to acquire, formalise, interlink and publish instructional knowledge on the web. This combined method is a possible solution to the know-how representation and know-how acquisition problems discussed in this thesis.

[17] `https://opendatacommons.org/licenses/` (accessed on 17/10/2017)

Chapter 7

Know-How Applications

Chapters 4, 5 and 6 describe, respectively, how instructional knowledge can be formalised; how it can be extracted from existing documents; and how it can be integrated with related data. The main goal of the whole process is to enable useful applications by generating instructional data which is 'more' machine-understandable. In this thesis, the level of machine understanding of a dataset is seen as a measurement of the number and *complexity* of the applications that we can *expect* a machine to implement given the dataset. An application can be said to be more complex than another if it allows a user to achieve a certain goal with a higher level of automation. Moreover, it can be said that a machine is expected to implement a certain application if there is a known method for developing such an implementation, or in other words, if it can be considered an engineering endeavour instead of a research project. These definitions are admittedly vague, as they depend on the informal concepts of complexity and expectation. Nevertheless, it captures the intuition that evidence of higher machine understandability of a formalisation can be provided by showing how it enables higher automation of certain goals which cannot be expected by data in the previous existing forms.

Given an HTML web page containing a description of a set of instructions, we can expect a machine to assist a user in a number of simple applications, which include displaying its content in a human-readable way, and retrieving additional documents by following the HTML links present in the document. If a user is interested in discovering further information about a set of instructions, in executing them, or in collaborating with other users, there is not significantly more automatic support that we can currently expect users to receive. The main goal of this chapter is to provide evidence that the proposed formalisation of instructions, the PROHOW model, is more machine-

	Centralised	*Decentralised*
Know-How Exploration	✓ (Section 6.1)	✓ (Section 6.2)
Know-How Execution	✓ (Section 6.3)	✓* (Section 6.4)

Figure 7.1: Overview of this chapter. The green check-mark denotes applications that I have implemented, and the grey starred check-mark denotes a conceptual framework that I have designed.

understandable. To do so, this chapter will describe two applications that cannot be efficiently implemented with the current HTML representation of instructions, but that are easily implementable using PROHOW data. These applications are derived from the ones that were considered in Section 4.7.3 during the formalisation of the competency questions of the model, namely know-how exploration and know-how execution. The first application that I will discuss is the use of instructional resources to support the exploration of instructional information by a user. The second application that will be considered is the use of instructional resources to support the execution of a set of instructions.

An overview of this chapter is given in Figure 7.1. I will present each application in two different scenarios, namely a centralised and a decentralised one. While centralised applications can reach high levels of efficiency, they also require a high-cost concentration of data and computational resources. I will present a decentralised version of these applications to demonstrate how PROHOW data can be stored on the web in a decentralised fashion while still supporting know-how exploration and execution. In other words, existing instructional resources on the web, which are to a large extent decentralised, can benefit from being formalised using the PROHOW model without requiring centralisation.

7.1 Know-How Exploration

The main use of instructions is to provide know-how to a user, or in other words, communicating information about how a task can be executed. A user interested in know-how might have various exploration goals, such as discovering the sub-tasks of a given task, discovering its requirements or comparing alternative ways to complete the same task.

7.1.1 Problem Description

I formulate the generic hypothesis that the PROHOW model allows users to reach several know-how exploration goals more efficiently than using the current representation of instructions as HTML documents. In this hypothesis, I am assuming the availability of PROHOW data derived from such documents. For each of these goals, the following more specific hypothesis will be considered:

Hypothesis 8. *The given know-how exploration goal can be achieved more efficiently using* PROHOW *instructional data compared to using the original* HTML *representations of instructions.*

The notions of efficiency and of exploration goals used in this section are the ones introduced in Section 3.3.1. The *original* representation of instructions refers to the web documents from which the PROHOW data was extracted from. The type of instructional web documents that I am considering are typical HTML representations of textual instructions, such as the ones available on the websites listed in Table 4.1 in Chapter 4.

It should be noted that all the features of the PROHOW model could, in theory, be reproduced without using the PROHOW vocabulary or RDF data. For example, all of the user interfaces that will be presented in this section could be reproduced by dynamic HTML pages generated from a relational database. In this scenario, however, it could be argued that the relational database is simply an alternative instantiation of the PROHOW model, or a model with equivalent expressiveness. The choice of using a Linked Data approach based on an RDF data model and URI identifiers is mostly a matter of practicality, as these standards are specifically designed to facilitate the integration of web data. Different equivalent representations are possible, although arguably their implementation would be impractical. In fact, many solutions already offered by Linked

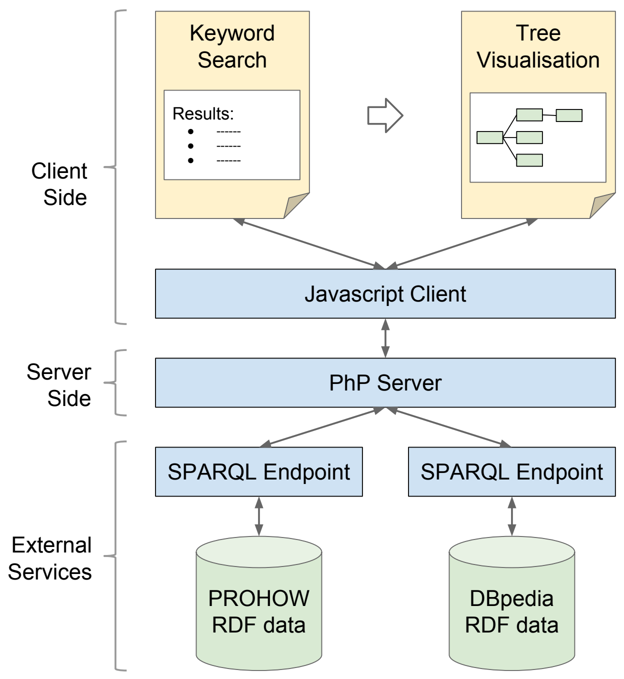


Figure 7.2: Architecture of the HowLinks tool.

Data technologies, such as universal identifiers, a common data model and shared vocabularies, would most likely need to be re-invented. Therefore, Hypothesis 8 should not be interpreted as a statement about the limitations of non-RDF resources, but as a statement about the benefits of having a richer structured representation that captures the semantic relations of the PROHOW model.

7.1.2 Experiments: the HowLinks Exploration Tool

In order to experimentally validate the formulated hypothesis for a number of exploration goals, a concrete know-how exploration tool called HowLinks was created. This tool has been developed by the author in collaboration with Benoit Testu at the National Informatics Institute in Tokyo, Japan.[1] HowLinks is a service implemented in PHP and Javascript that offers an integrated visualisation in a web browser of PROHOW and DBpedia data accessed through SPARQL endpoints. The source code for this application has been published on GitHub.[2]

The architecture of the HowLinks tool is summarised in Figure 7.2. A Javascript client displays HTML pages to the users by retrieving data from a PHP server-side script. The server-side script answers to the requests of the client by collecting data

[1] `http://ri-www.nii.ac.jp/HowLinks/index.html` (accessed on 17/10/2017)
[2] `https://github.com/paolo7/howlinks` (accessed on 17/10/2017)

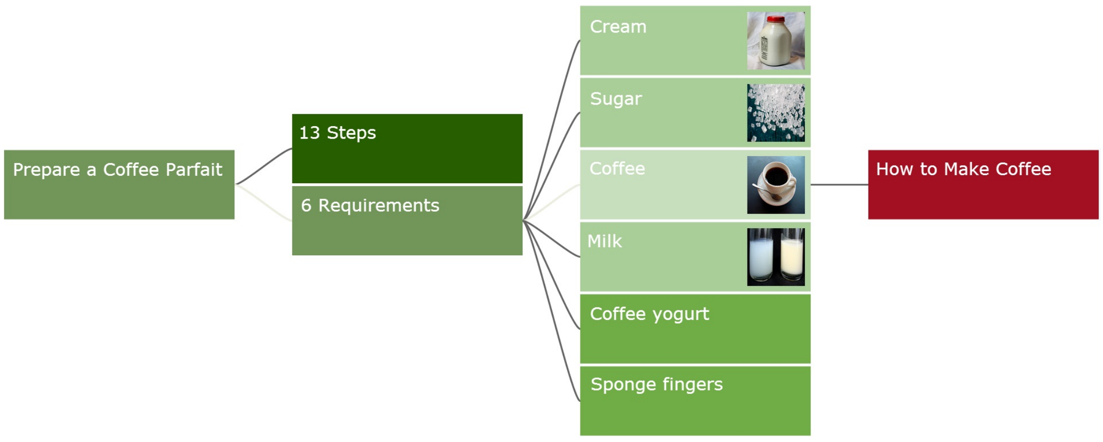


Figure 7.3: Screenshot of the HowLinks web application.

from a number of SPARQL endpoints. Each instance of HowLinks is configured with up to two SPARQL endpoints: an endpoint exposing a PROHOW dataset and an optional endpoint exposing a DBpedia dataset. Know-how exploration in HowLinks starts with a keyword search. The keywords used in the search are transformed into a SPARQL query that retrieves the URIs of the tasks that contain such keywords in their label. To limit the amount of results to the most meaningful ones, this search is not performed over all tasks, but only over the top-level tasks of each set of instructions. The labels of the tasks returned by the query are displayed to the user in a list, and he or she can then select one of them to begin the exploration.

Once the user has selected a specific task, queries are performed to retrieve information about that task. More specifically a set of queries returns information about (1) its steps, (2) its methods, (3) its outputs, (4) its requirements, (5) its related DBpedia information and (6) the tasks that require such object if it represents the output object of a process. With the exception of DBpedia information, all the other pieces of information are information about tasks. These tasks are displayed in a tree structure as children nodes of the original task, and if they are above a certain number, they are grouped together into a single node. Each node in this tree structure represents a task or a collection of tasks, and is colour-coded depending on its relation with its parent node. For example, steps are represented with the colour orange while requirements with the colour green. Each node representing a collection of tasks can be expanded into its members. Each task node displays the label of the task. Moreover, if the task involves obtaining or producing an object associated with a DBpedia resource type, the picture of that type obtained from DBpedia will also be displayed. Clicking on a node will expand it with its children.

Figure 7.3 displays a partial screenshot of the visual output of HowLinks. In this figure, a user has expanded information on how to “Prepare a Coffee Parfait” after discovering this task using the keywords “coffee” and “parfait”. The top level node of the tree represents this task, and its children can be expanded to discover its steps or its requirements. In this figure, the user has expanded its 6 requirements, 4 of which are linked to DBpedia types and therefore include a picture retrieved from DBpedia. Finally, the user has expanded the requirement “Coffee” and discovered the task “How to Make Coffee” which can be used to generate such requirement. The user could now expand this task to discover more information on how to make coffee, or he or she could continue the exploration by clicking on other nodes.

7.1.3 Evaluation of Know-How Exploration Goals

I will now formulate a number of exploration goals that the HowLinks tool enables users to achieve efficiently using the PROHOW representation of instructions. They will be compared with the wikiHow and Snapguide websites[3] to evaluate to what extent the same goals can be achieved on these websites. Following the methodology described in Section 3.3.1, I will evaluate the efficiency of achieving a goal based on the set of actions necessary to achieve it. The four types of actions I will consider are the following:

Click A click (shortened as `c`) is an action achieved by a user by clicking an element of the interface.

Page Scroll A page scroll (shortened as `ps`) is an action achieved by a user by scrolling through a list of elements across one or multiple pages.

Web Search A web search (shortened as `ws`) is an action achieved by a user by retrieving a resources that describes an object using a search engine.

Complete Instructional Search A complete instructional search (shortened as `cis`) is an action achieved by a user by searching through all of the sets of instructions in a repository to look for specific types of information.

Given the impracticality of performing a `cis`, a system that requires it to achieve a goal is always considered less efficient than one that does not require it.

[3]The version of these websites available in April 2017.

Know-How Exploration Goal	wikiHow	Snapguide	HowLinks	KnowHow4j
Output Process Discovery	c / cis	c + ps	c	cis
Common Input Discovery	cis	c + ps	c	cis
Decomposition Discovery	c / cis	cis	c	c
Integrated View Discovery	np	np	c	c
Related Factual Knowledge Discovery	c + ws	c + ws	c	c + ws

Table 7.1: The actions required by different systems to achieve the five exploration goals. A goal which is not achievable by a system is labelled np. The plus symbol + denotes that both actions are necessary, while the slash symbol / denotes that one of the two actions is necessary.

The five exploration goals that will be considered are listed in Table 7.1. This table compares HowLinks to wikiHow and Snapguide with respect to the level of support of these goals. This table also includes the level of support of these goals by the KnowHow4j tool, introduced in Section 5.6. The decentralised know-how exploration functionality offered by KnowHow4j will be discussed later in Section 7.2.

Output Process and Common Input Discovery Goals

By expanding a requirement task of a process in HowLinks with a single click (c), a user can achieve the two exploration goals, namely *output process discovery* and *common input discovery*. The output process discovery involves discovering how a requirement of a set of instructions can be obtained. Arguably, this goal is of interest to many users, such as those which do not already posses a requirement, and might consider how difficult it would be to create it. For example, a user inspecting a set of instructions that require the ingredient “Pancake” might be interested in discovering the output process “How to Make a Pancake”.

The common input discovery goal involves discovering other tasks that require the same type of input. Discovering other tasks that can be accomplished with a certain input has many uses, such as estimating the usefulness of a tool, or discovering more recipes that contain the user’s favourite ingredient. For example, a user that wants to use the ingredient “chocolate” might be interested in finding other sets of instructions that require this ingredient, such as “How to Make a Chocolate Cake”.

In wikiHow, requirements of a set of instructions can be linked to at most one set of instructions that explains how to obtain them.[4] If only one such set of instructions exists, then it can be accessed with a single click (`c`). However, if multiple such sets of instructions exist, a complete instructional search (`cis`) would be required to find them. Moreover, wikiHow does not support the common input discovery process, as the only way to discover other sets of instructions that involve the same requirement is through a complete instructional search (`cis`).

In Snapguide, requirements of a set of instructions are linked to a page that lists other sets of instructions that either require or produce objects of the same type.[5] These two different types of sets of instructions are mixed together. Therefore, a user that is only interested in one of the two types, needs to first access this page (`c`), and then to scroll through the list of results to find the ones he or she is interested in (`ps`).

Decomposition Discovery Goals

This *decomposition discovery* goal involves discovering tasks that can be used to decompose a step. This is a common goal, as tasks are often not described in details, and certain users might need to gather more information about a task from external sources in order to properly understand it. Using the HowLinks tool, a user can achieve this goal by expanding the steps of a process with a single click (`c`).

In wikiHow, the HTML description of steps can contain a link to another set of instructions that explains how to achieve that step or part of it. This link is located in the textual part of the label of the task that describes the part that is related to the external set of instructions. For example, the words "prepare a pancake" in the label of a step: "Next, prepare a pancake for every guest" could contain an HTML link to the set of instructions on "How to Make a Pancake". This approach, however, can only represent one such link for every textual component of the task, and multiple relevant tasks, such as multiple pancake recipes, cannot be linked. Therefore wikiHow allows the discovery of one possible decomposition with one click (`c`), while a complete search might be necessary to discover alternative decompositions (`cis`).

In Snapguide, similar decomposition links to the ones available in wikiHow would be possible, but none was found. Therefore, a complete instructional search is required to achieve the decomposition discovery goal (`cis`). Overall, representing decomposition links as HTML links within the text of a task limits the number of decomposition tasks that it is possible to link to. While it is true that multiple links for the same com-

[4]For example: `https://www.wikihow.com/Fondant-a-Cake`, accessed on 14/04/2017.
[5]For example: `https://snapguide.com/supplies/fondant/`, accessed on 14/04/2017.

ponent could be represented using other HTML structures, such as in a list, or in a pop-up display, adding such structures can be considered as imitating certain HowLinks functionalities.

Integrated View Discovery Goals

Users might also be interested in visualising sets of instructions on the same page to see how they relate with each other. For example, a user following a cooking recipe on how to make a pancake might want to expand the step "prepare pancake mix" with its sub-steps, as described in another set of instructions, without having to navigate to a different page. This objective can be called the *integrated view discovery* goal.

In wikiHow and Snapguide, sets of instructions can only be visualised as a whole and there is no option for combining related ones. This goal is achievable by HowLinks, which is designed to provide an integrated visualisation of multiple sets of instructions, or relevant subsets of them. In fact, by clicking on a component of a set of instructions (C) information about this component is displayed in the same interface, regardless of which set of instructions it was originally part of. Achieving this goal is a direct result of an advantage of using a structured representation of instructions. In a similar way as in a modelviewcontroller architecture, the PROHOW representation of know-how is decoupled from the visual environment where it is presented to the users. In this scenario, a simple set of display rules could be sufficient to visualise any set of tasks originating from different sets of instructions.

Related Factual Knowledge Discovery

Users that are not only interested in know-how but also in related factual knowledge can be said to pursue a *related factual knowledge discovery* goal. For example, a user looking at a recipe involving an exotic ingredient might be interested in discovering what this ingredient is. To the best of the author's knowledge, wikiHow and Snapguide do not support this goal. Using these websites, a user would need to click on a different interface (C) and then perform a web search (WS) in order to discover more information on a requirement.

HowLinks, on the other hand, allows users to discover information from DBpedia related to the task at hand. In its current implementation, HowLinks displays a picture taken from DBpedia to complement the textual description of a task with a visualisation of its type. Given that the URI of the related DBpedia resource is known, it would also be trivial to display additional information, such as a textual description, if the user clicks on the picture (c).

7.1.4 Discussion

In this section I have presented five know-how exploration goals and discussed how efficiently they can be achieved using the HowLinks tool instead of the wikiHow and Snapguide websites. The results of the analysis presented in Table 7.1 show that users using HowLinks can achieve all the exploration goals by performing one click. Achieving the same goals using the other wikiHow and Snapguide websites, however, is either less efficient or not possible at all. This can be taken as evidence that Hypothesis 8 holds true for the five goals that have been considered. This result can be attributed to a number of advantages brought by the PROHOW formalisation.

As discussed before, one of the main advantages of PROHOW, and of many structured representations, is decoupling information from its visual representation. This decoupling also allows for RDF links between entities to be discoverable regardless of their directionality once they are stored in a single database. I use the term RDF *link* to refer to edge of the directed labelled graph interpretation of an RDF dataset. HTML links, on the other hand, can only be traversed in one direction. An HTML link between resource *A* and resource *B* allows the discovery of *B* from *A* but not *A* from *B*. HTML links can be made to link two resources bi-directionally by providing a link from each resource to the other one, though this requires link duplication. Instructional websites often contain links between a step and instructions on how to accomplish it. The reverse links, however, are typically not present as they would be expensive to maintain.

Moreover, the cost of representing many-to-many relations between documents using HTML links grows exponentially. For example, an instructional dataset might contain 10 recipes to prepare pancake batter that can be used by 10 instructions on how to prepare pancakes. If we want to represent all the I/O links between these instructions we will need 100 HTML links. In the PROHOW model instead, the entity representing the common object type, in this case "pancake batter", acts like a central node to which the I/O links are directed to and from. The I/O links of this example, therefore, can be represented using only 20 RDF links.

Finally, RDF links are labelled, and are therefore more informative than HTML links. This allows systems using PROHOW data to filter the set of resources related to a certain task based on the type of relation that a user is interested in. For example, it is known whether a resource B linked to resource A provides information on how to achieve A, or about what type of object A produces.

7.2 Decentralised Know-How Exploration

The knowledge exploration tool described in the previous section, HowLinks, demonstrates how it is possible to provide an integrated visualisation of a large dataset of PROHOW instructions. The main drawback of this tool, however, is the necessity to store a large amount of RDF data within a single repository. This can be considered a major disadvantage compared to HTML instructions, which can be created and interlinked by any web user with relatively little cost. Although centralised instructional repositories such as wikiHow and Snapguide are common, any web user has the possibility of creating an independent set of instructions, hosting it on a different website, and then linking it to a set of instructions in a centralised repository. A structured representation of instructions should offer to the users "the best of both worlds", and not restrict the users' ability to create and use knowledge. The goal of this section is to demonstrate how PROHOW data can be created and used in a decentralised fashion in the context of know-how exploration. I formulate this goal as the following hypothesis:

Hypothesis 9. *The* PROHOW *representation of instructions embedded in* RDF*a in decentralised* HTML *pages can be integrated and explored on-demand if a* URI *naming convention is used. In particular, this naming convention should enable the discovery of the* URI *of the main task described by the set of instructions from the* URL *of where the* HTML *page is stored.*

7.2.1 Problem Description

In Section 5.6 I presented the editor functionality of the KnowHow4j[6] tool, a Javascript tool that I developed to generate and visualise PROHOW representations of instructions using RDFa. The editor functionality of this tool allows the user to define PROHOW instructions as a set of steps, methods and requirements. The resulting PROHOW data is translated into an HTML code snippet containing both the human-readable set of instructions in plain HTML and the machine readable PROHOW data in the RDFa format.

[6]`https://github.com/paolo7/khjsdevelop` (accessed on 17/10/2017)

While this approach can easily be used to publish isolated, PROHOW-embedded, web documents, allowing explorable RDF links between them requires additional consideration. More specifically, when creating a new set of instructions A that describes a set of tasks T, two problems arise. The first problem arises when trying to link a step $t \in T$ to a task x described in a separate set of instructions B. This problem is the discovery of the URI of x. Without knowledge of the URIs of both t and x, a link between them cannot be established in RDF. The second problem arises when exploring the know-how contained in A. This problem is the discovery of the related PROHOW resources in B. Without gaining access to the external set of instructions B, it is not possible to exploit the available knowledge of task x.

As an example scenario, we can imagine a user creating a set of instructions about task a: “How to prepare a pancake” which involves the step b: “Prepare the pancake mix” that he or she would like to link to an external set of instructions about task c: “How to prepare a pancake mix”. While the user might not know the URIs of all the entities in each set of instructions, I assume that he or she knows the URL of the external set of instructions describing task c. A possible method to solve this problem will be presented in the following section.

7.2.2 A Method for Decentralised Know-How Exploration

I will now present a method to solve the two main problems of decentralised know-how exploration. Following the methodology presented in Section 3.3.2, I define the three necessary requirements of this method:

RDFa embedded HTML pages This method assumes that web resources describing instructions are represented as HTML pages embedded with RDF data using RDFa annotations.

PROHOW annotations of instructions This method assumes that instructions are accompanied by RDF data that uses the PROHOW vocabulary to describe the instructions.

A URI naming convention A naming convention is used to allow a system to infer the URI of the top-level task described in a resource from its URL. This convention will be described in more details in Section 7.2.2.1.

In Section 7.2.2.2 I will propose a method to dynamically discover and integrate

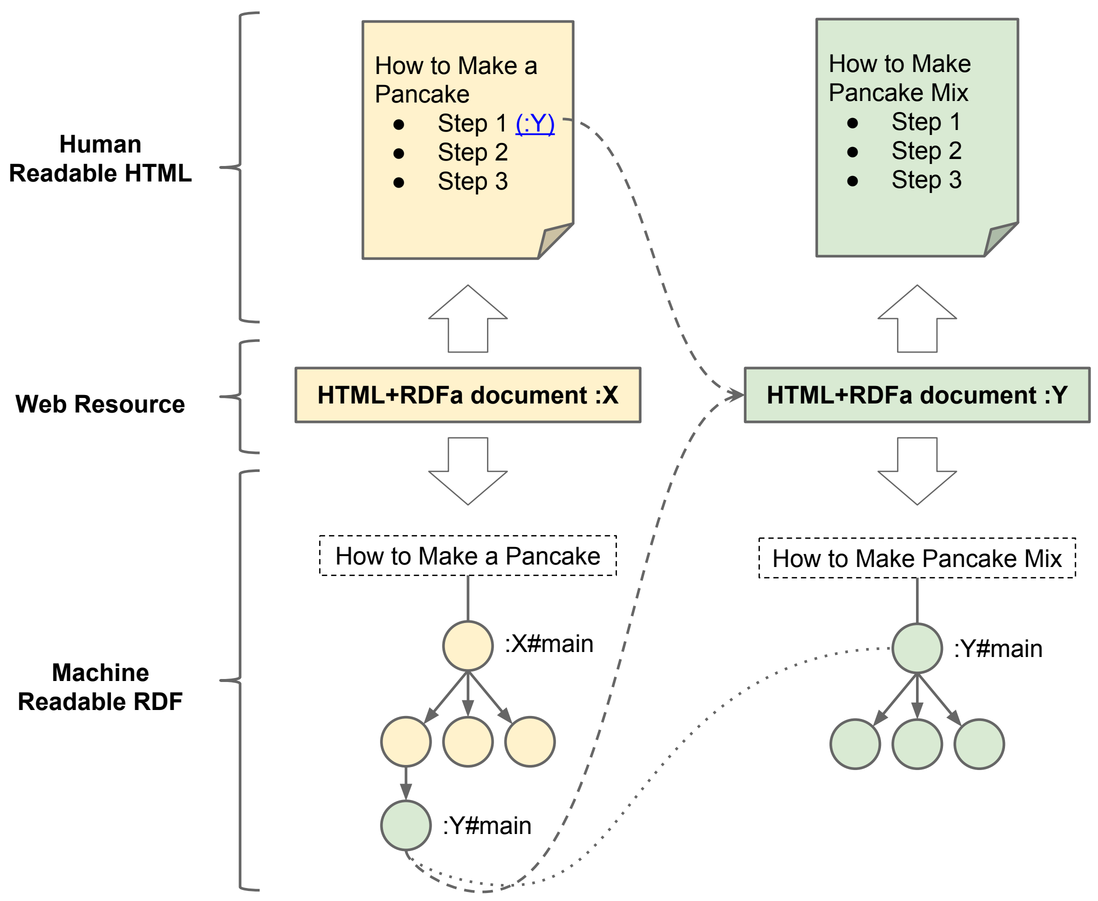


Figure 7.4: Schema of the HTML and RDF links in resources generated by the KnowHow4j tool.

resources that follows the above mentioned requirements using dynamic RDF discovery.

7.2.2.1 URI Naming Convention

The first problem that I will address is the discovery of the URI of the task to link to. Since URLs are a type of URIs, a user interested in creating a link to an external set of instructions can be assumed to know at least one URI associated with this set of instructions. This URI refers to the page that contains the instructions and it would not be semantically correct to re-use the same URI to denote one of the tasks it describes. Instead, I propose a standard convention to denote the top level task defined in the set of instructions. The top-level task, as previously defined in Section 6.4, is the main task that the set of instructions describe. The top level task can be identified by appending a *fragment identifier*[7] to the URI of the page that describes it. A fragment identifier is a

[7]`https://tools.ietf.org/html/rfc3986#section-3.5` (accessed on 17/10/2017)

short string of characters starting with the hash symbol "#". Fragment identifiers are an optional part of a URL that do not change the resource that the URL points to. In Linked Data, it is a standard practice to use fragment identifiers to create multiple URIs, called *hash* URIs, that can be used to retrieve the same web resource while having a different URI from it (Heath et al. 2011).

This method is currently used only to identify the top level task of a set of instructions. Although it would be possible to create other conventions to identify other components of an external set of instructions, such as "the first step", there is no guarantee that such components exist. The only component that it is assumed to be present in any set of instructions is the top level task. As presented in Section 6.4, top level tasks can be linked to bottom level tasks to create decomposition links across sets of instructions.

The URL of the resource containing the set of instructions does not need to be specified in advance, as in RDFa it is possible to use *relative* URLs. When retrieving a web resource, relative URLs are automatically instantiated with the *base* URL of their context. Unless special configurations are used, the base URL typically refers to the URL of the web resource that is being requested by a client, and it is prefixed to the relative URLs. For example, the relative URL `#task` found in a web resource with URL `http://example.com` would be instantiated as `http://example.com#task`.

The advantage of using relative URLs compared to absolute ones is that the user does not need to specify in advance where a resource will be published. Moreover, the same document can be moved to a different URL without the need to change its underlying RDFa data. The main disadvantage, however, is that by moving the location of this document its internal URIs will also change. This means that if an external dataset contained a link to one of these URIs, after the resource is moved this URI might no longer be valid. Although using absolute URLs guarantee the permanence of the URIs used in RDFa, this is not always an advantage. In fact, it will be discussed next how maintaining an absolute URI after moving the web resource that describes it can create problems in dynamic RDF discovery.

7.2.2.2 Dynamic RDF Discovery

Having established a convention for identifying the URI of the top level task of an external set of instructions, the remaining problem is how to retrieve additional information about this task. This problem can be considered already solved by having the the URI of the top level task also function as a URL capable of retrieving information about the task. The URI of a top level task defined using this method, in fact, points by default to the web resource that contains its PROHOW description embedded in RDFa. When information about a task *x* is sought, it is possible to *dereference* this URI and retrieve RDF data stored at this URL. If the resource retrieved is an HTML page, its contents will be parsed to extract the embedded RDFa data. The data extracted is then added to the local RDF dataset stored in memory, which can be later visualised. This method is also compatible with RDF formats other than RDFa, such as Turtle and RDF/XML.

The type of URIs described in this section can be considered HTTP URI*s*, namely URIs that point to web resources retrievable through the HTTP protocol. Since the web resources retrieved by one such HTTP URI contains an RDF description of this URI along with other HTTP URIs, this data can be said to be following the Linked Data principles. The interlinked RDF resources generated by this method can be considered a type of Linked Data and their exploration as dynamic Linked Data discovery. In the example presented in Figure 7.4, both URL `:Y` and URI `:Y#main` point to the same web resource. Therefore, resource `:Y` is discoverable from resource `:X` both by following an HTML link and by performing dynamic Linked Data discovery.

It should be noted that this method for generating and retrieving RDF resources dynamically is best suited for resources with a static location. If the resource containing the PROHOW representation of a task is relocated to a different URL, its URI will change and all of the external occurrences of its previous URI should be updated accordingly. This is an inherent problem of URL resources and HTML links are subject to the similar risk of “breaking” if the resource they point to is relocated. If there is a high risk that a resource will need to be moved, a possible solution to this problem is to use URL redirection systems such as Persistent Uniform Resource Locators (Weibel et al. 1996).

7.2.3 Experiments: The KnowHow4j Exploration Tool for Decentralised Resources

I have developed an implementation of this method as part of the KnowHow4j tool, using the rdflib.js[8] Javascript library to parse and query RDF resources such as RDFa embedded HTML files and Turtle data files. The specific URI naming convention used by the KnowHow4j tool is to append the text fragment `#main` to the end of a URL of a web page to generate the URI of its top-level task.

[8]`https://github.com/linkeddata/rdflib.js/` (accessed on 17/10/2017)

In the KnowHow4j editor, users can specify any number of links to external web pages from any component of the set of instructions. These links are further described as one of three PROHOW relations, namely `prohow:requires`, a `prohow:has_step` or a `prohow:has_method` link. To specify this information, users need to write one of the keywords: "requirement"', "step" or "method", and the URL to link to, within square brackets at the end of the description of a component. When parsing this set of instructions, the KnowHow4j editor will automatically add the fragment `#main` to the end of the linked URLs.

Similarly, when creating a set of instructions, the KnowHow4j editor will create the URI of the top-level task by appending the fragment `#main` to the URL of the document where the HTML code will be published. Figure 7.4 illustrates the main components of two interlinked sets of instructions generated with the KnowHow4j editor. These resources are published on the web at URLs `:X` and `:Y` and the top level task described in these resources is identified, respectively, with URIs `:X#main` and `:Y#main`.

The KnowHow4j tool includes a visualisation functionality. This functionality is accessed by opening its HTML interface on a web browser. From this interface, a user can start exploring PROHOW resources by providing the URL of a resource containing PROHOW data. As mentioned before, this resource could be dereferenced into an RDFa embedded HTML page or more generally, any serialisation of an RDF data file.

After providing the initial URI of a task, KnowHow4j will attempt to interpret the URI as a URL and retrieve the resource it points to. If successful, the resource will be parsed and all the discovered RDF will be loaded in memory locally. If a PROHOW description of the task is found, it will be displayed graphically as an expandable tree in an equivalent fashion to the one described before for the HowLinks tool.

The main difference between the KnowHow4j and HowLinks visualisation tools is the effect of expanding the nodes of the graphical tree representation. In KnowHow4j, expanding a node in the tree that represents an entity with a certain URI results in an attempt to retrieve more RDF data dynamically from this URI. If successful, this additional data will be added to the local dataset stored in memory and it will be used to display more information about the expanded task.

7.2.4 Discussion

I have proposed a method to automatically explore and integrate distributed instructional resources if they fulfil the three requirements listed in Section 7.2.2. I have tested this method empirically by developing a prototype: the KnowHow4j application. This application and a detailed tutorial on how to replicate the discussed functionalities is available on the tool's GitHub's page.[9] As listed in Table 7.1, KnowHow4j allows a user to achieve two of the exploration goals defined for the HowLinks application in Section 7.1.3, namely decomposition discovery and the integrated view discovery goals. The decomposition discovery is achieved by clicking on a task to expand it into its sub-tasks. This goal is achievable under the assumption that a link between a bottom level task x and a top level task y is stored in the same resource that describes x. By dynamically retrieving and integrating information coming from different sets of instructions, this tool also achieves the integrated view discovery goal.

The decentralisation approach taken by KnowHow4j also results in certain drawbacks. Decentralised resources need to be discovered and retrieved before they can be used, for example in a visualisation. The approach used by KnowHow4j to discover decentralised resources from an initial RDF dataset results in three constraints. The first constraint is that URIs are not simply used as semantic identifiers, but they also need to function as URLs pointing to a relevant web resource.

The second constraint is that the RDF statements contained in decentralised resources need to enable exploration. In other words, if a resource B needs to be discoverable from a resource A, then a URI that points to B needs to be found in A. The directionality of resource discovery results in the fact that although two resources A and B are related, only one of them might enable the discovery of the other. It is possible, therefore, that related information exists but it cannot be discovered. This is not a problem in the centralised HowLinks applications because all of the relevant datasets are assumed to be known.

The third and last constraint is that instructional documents need to follow a set of publication conventions, such as publishing documents with embedded RDFa data, and following certain URI naming conventions. This can be considered as a more restrictive publication approach than the one currently used in practice, as real-world instructional articles can be published without having to follow specific conventions. However, I argue that this is not a severe limitation, as compliance with the required publication conventions can be ensured by using simple editing tools, such as KnowHow4j.

This application demonstrates it is possible to navigate a decentralised network of human and machine-understandable instructional resources, thus validating Hypothesis 9. The exploration of this network does not require, *in theory*, any server-side func-

[9] `https://github.com/paolo7/khjsdevelop` (accessed on 17/10/2017)

tionalities or any specialised software other than a modern web browser. In practice, however, modern web browsers do not allow scripts to dynamically access resources from different domains under the *same-origin policy*.[10] Different ways to circumvent this policy exist, and the KnowHow4j tool uses one such approach that relies on a proxy service.

7.3 Know-How Execution

Having discussed ways to enhance the exploration of instructional resources, I will now focus on ways to support their execution. With an increase in the capabilities of artificial systems, the number of everyday tasks that involve a mixture of human actions and machine computation is also increasing. In this section I investigate the extent to which the PROHOW formalisation of instructions allows machines to support the human execution of those instructions. I will present the first framework that allows non-programmers to create and execute workflows where each task can be completed by a human or a machine. In this framework, the PROHOW dataset represents a shared knowledge base that humans and machines use to interact with each other. The hypothesis that will be considered is that non-programmers can describe how to achieve certain tasks at a level of abstraction which is both human and machine-understandable. Going back to the initial goal mentioned at the beginning of this chapter, I will provide evidence of an increased level of machine understanding of a task by automating certain steps of the task that were initially intended to be executed manually. The contents of this section have been published as a conference paper (Pareti et al. 2016).

7.3.1 Introduction to the Human-Machine Collaboration Scenario

This section addresses the 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[11] (If-This-Then-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”.

[10] `https://tools.ietf.org/html/rfc6454` (accessed on 17/10/2017)
[11] `http://ifttt.com/` (accessed on 17/10/2017)

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. One of the main disadvantages of human instructions, however, 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.

The main contribution presented in this section is a framework that allows non-programmers to define and execute human-machine workflows. A human-machine workflow is a workflow composed by tasks that can be potentially completed both by humans or by machines. This approach addresses the limitations of traditional human instructions by allowing machines to automatically detect when certain actions are needed, and consequently to actively execute them. Section 7.3.4 presents the components of this framework and describes how they achieve its objective. An implementation of this framework is described in Section 7.3.5 and its usability by non-programmer users is evaluated in Section 7.3.6.

7.3.2 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 *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></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 has programming skills. Consequently, although some of the required functionalities are available in their computers, the whole task might have to be completed manually.

The proposed solution for Jane's problem is to let her use a system that automatically translates her natural language instructions into a formalised representation, such as PROHOW, 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 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 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, I propose an approach to automation that 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, his or her 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.

7.3.3 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 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 project. To evaluate whether this different level of abstraction is a solvable problem, I formulate the following hypothesis:

Hypothesis 10. *There is a non-empty intersection T between the set H of tasks that can be* easily defined *by (non-programmer) humans, and the set M of those that are automatable by machines.*

Lacking a previous baseline to compare to, we can consider 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. A task will be considered *defined* if a user can generate a PROHOW representation of the task which includes, for example, its steps and requirements.

The objective of this work 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 will be addressed is how to allow a (non-programmer) web user to share (1) human-made instructions and (2) information on the progress

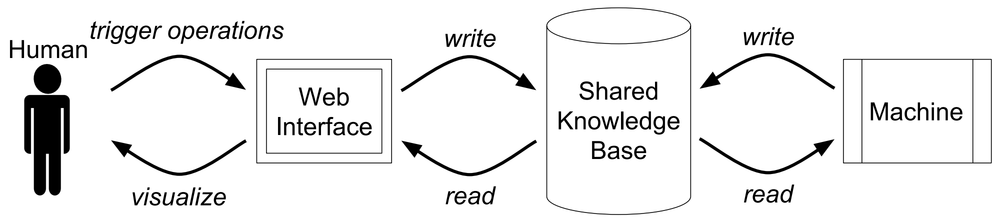


Figure 7.5: Schema of the main components of the human-machine collaboration framework. Humans and machines interact with each other through a shared knowledge base.

made at completing the tasks described in those instructions, in a form that is both human and machine-understandable.

7.3.4 A Method for Human-Machine Collaboration

At the core of the proposed approach for 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 Figure 7.5. This knowledge base adopts the PROHOW vocabulary to represent know-how and its execution.

In this approach, communication is performed indirectly, by modifying the shared knowledge base. This can be seen as a type of *stigmergy* (Omicini et al. 2004), 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 dataset. 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.

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 form. 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 the PROHOW model, 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 Section 7.3.5.

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. The typical workflow of the proposed framework will now be described across these phases.

7.3.4.1 Know-How Acquisition

In the first phase of this framework, know-how is converted into a PROHOW representation. This conversion is based on the assumption, validated in Chapter 5, that humans can write instructions in a semi-structured form, such as the one of wikiHow instruction pages. For example, we can consider the following natural language instructions for the string transformation task `:t` described in Section 7.3.2:

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>.

By exploiting the implicit structure of this text, it is possible to automatically generate the following RDF graph:

```
: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 .
```

7.3.4.2 Know-How Linking

In the second phase of this method, 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`.

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 Chapter 6 I presented a fully automated link discovery system. In this chapter, instead, I will consider a semi-automated approach. Whenever a user creates a set of instructions, an integration service 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.

7.3.4.3 Collaborative 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. In the PROHOW vocabulary, the fact that the execution `:ex1` of the first step `:t1` of the instructions `:t` has been completed can be represented with the following graph:

```
:ex1 prohow:has_task :t1 .
: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.

7.3.5 A Human Interface 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, one such interface has been developed as an online service[12] which allows users to follow the three main phases defined in Section 7.3.4. This interface includes an online editor that translates the user's input into PROHOW data in a similar fashion than the KnowHow4j tool. 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. The graphical interface also allows users 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.

7.3.6 Experiments

The objective of this experiment is to support the hypothesis that the majority of (non-programmer) Web users can define certain tasks at a level of abstraction which is understandable by machines. This hypothesis is evaluated with respect to the three phases of the framework described in Section 7.3.4: know-how acquisition, know-how linking and execution. This experiment is also meant to demonstrate the application of

[12]`http://w3id.org/prohow/editor` (accessed on 17/10/2017)

this approach in a concrete scenario. Human participation in this experiment has been obtained through the crowdsourcing platform Crowdflower.[13] Participants were asked information about their computer skills to exclude answers from computer experts.

7.3.6.1 Evaluation of the Know-How Acquisition Phase

The hypothesis relevant to the know-how acquisition phase is that humans can write semi-structured procedures which can be automatically parsed into an RDF representation. This was evaluated by asking 10 workers to solve the example task described in Section 7.3.2 through an online survey.[14]

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.[15] Of the 10 submissions received, 3 were rejected as non-genuine attempts. Of the remaining submissions, the best 5 were considered for the next part of the experiment.

The number of workers involved in this first part of the experiment was kept small due to practical constraints. One constraint comes from the fact that collecting creative contributions is a notoriously difficult task using crowdsourcing techniques, due to the difficulty in automatically validating the user's contributions (Eickhoff and de Vries 2013). Another constraint comes from the fact that the more sets of instructions are considered, the more potential links to automatable functions need to be evaluated in the subsequent phase of the experiment. The main disadvantage of considering a low number of sets of instructions is that it reduces the statistical significance of this evaluation. However, as discussed in Section 3.3.3, the value of this experiment comes in demonstrating the feasibility of a know-how execution system in at least one scenario. The extent to which these findings can be generalised, instead, is not investigated in depth.

7.3.6.2 Evaluation of the Know-How Linking Phase

To judge whether non-programmer humans can correctly link instructions to automatable functions, each of the 16 steps of the five best sets of instructions was paired with 10 different machine functionalities. All automatable steps have been paired with one

[13]`http://www.crowdflower.com/` (accessed on 17/10/2017)
[14]`http://w3id.org/prohow/r1/survey` (accessed on 17/10/2017)
[15]`http://w3id.org/prohow/r1/survey_results` (accessed on 17/10/2017)

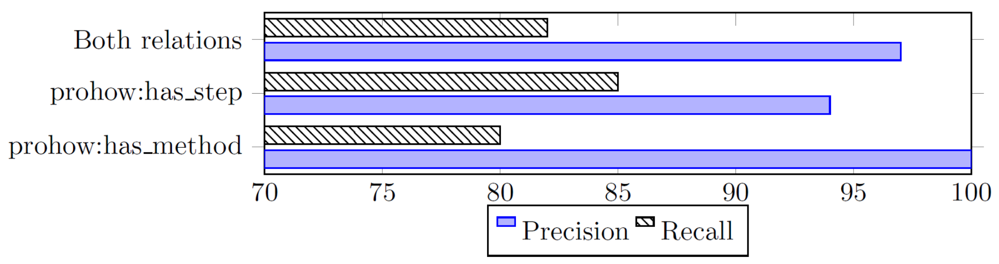


Figure 7.6: Precision and recall of the links generated by the workers.

or more relevant functionalities, as well as unrelated ones. For each step-functionality pair, workers were asked to judge whether a particular functionality (in the survey called *action*), such as "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 the judgement of 10 different workers was obtained, and then the judgement given by the majority of the workers was selected. 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, the first answer is interpreted 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. The links generated by the majority of the workers were manually evaluated and the precision and recall of those links is shown in Figure 7.6. The result of this evaluation shows that in this experiment 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.

7.3.6.3 Demonstration of the Execution Phase

To enable collaborative human-machine execution, the machine functionalities which workers previously created links to have been implemented. The system that I developed 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,[16] and each of them contains at least one automatable function.

7.3.7 Related Work on Human-Machine Collaboration

This know-how execution application allows humans and machines to collaborate by interleaving human and machine actions to achieve a common goal. This application is related to a set of partially-overlapping research areas focussed on mixed human and machine computations which have been reviewed by Quinn and Bederson (2011). Human Computation (HC) is one of these areas, and it is focussed on combining human and machine efforts to solve tasks that neither humans nor machines alone could solve efficiently. However in typical HC systems, such as Galaxy Zoo,[17] humans play a subordinate role, and their collaboration relies on an existing centralised software system. More control is given to users of Human-Provided Services (HPS) (Schall 2012). In HPS, users can define and advertise the services they want to offer, although they are not in charge of the overall computation.

Related to HC are the fields of Social Machines (SM) and Social Computation (SC) (Shadbolt et al. 2013). While HC systems might not have a social dimension, SM and SC focus on how to support social interactions between users. A relevant SC system is LSCitter, created by Murray-Rust and Robertson (2014). LSCitter provides automatic support to users performing collaborative tasks, such as organising a meal or sharing a taxi. While these collaborations are initiated by the users, they must currently follow pre-defined protocols. LSCitter is based on the Lightweight Social Calculus (LSC), an extension of the Lightweight Coordination Calculus (LCC) developed by Robertson (2005). LSC is a method to specify the protocols of interactions between multiple human and machine agents and to enable these interactions computationally in a distributed fashion.

The WeDo system, developed by Zhang et al. (2014), also aims to support human collaborative activities. WeDo provides this support by acting as a coordinator between the human participants, to collect knowledge on how to achieve a task first, and to organise its execution later. The coordination support provided by WeDo, however, cannot be customised by the users.

In all of these areas, expert intervention is required to define and customise the

[16]`http://w3id.org/prohow/r1/instructions` (accessed on 17/10/2017)
[17]`http://www.galaxyzoo.org/` (accessed on 17/10/2017)

human-machine collaboration workflow. As a result of this, users do not currently have freedom to define how to achieve their individual goals. Compared to these research areas, my know-how execution application features one major novelty. Users of my application are in charge of defining the human-machine collaboration workflow that they want to follow and are in control of initiating and managing the computation. An additional feature of my application that is not typically found in other systems is that automation can happen opportunistically, as any task could in principle be accomplished manually. In other words, in different environments, the same set of instructions could be automated to a lesser or greater degree.

7.3.8 Discussion

In this section I 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, I 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 dataset using the 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.

The main objective of this work is not to describe how humans and machines collaborate, but rather to enable this collaboration in practice. To demonstrate this, I presented an implementation of the proposed collaboration framework which was evaluated in a concrete test scenario. The results of this experiment support the hypothesis that non-programmers can specify certain types of instructions at the level of detail required for machine understanding. However, these results have been obtained by considering only a single scenario. The extent to which they can generalise to other scenarios and domains has not been investigated.

This hypothesis is related to the previous pieces of work described in Section 2.3.3, which are focussed on learning automatable commands from non-programmer users.

These pieces of work, however, focus on complete automation and are not intended to support human-machine collaboration, where tasks are only semi-automatable. Although this previous work does not make use of the PROHOW model, it can be seen as further evidence of the ability of non-programmer users to define reusable executable commands when supported by effective procedural-knowledge capturing tools. An interesting avenue for further research might involve the integration of such approaches within the second phase of the framework described in Section 7.3.4.2 to link a set of instructions to machine functionalities.

The machine functionalities discussed in this chapter could also be implemented as services, such as Web Services, which represent reusable automated functions that can be requested when needed. For example, a weather service can be invoked to discover the weather conditions in a specific location. Their integration with human instructions can then be seen as a form of service composition. An approach to discover and compose services exploiting the natural language description of services has been proposed by Sangers et al. (2013). Sangers et al. make an important point about the need for services to be easily understandable by non-technical people, which is also relevant in my work. They argue that the composition of services should be driven by those which understand their semantic properties, such as business experts, rather than those that understand their syntactic properties, such as technicians with service composition skills. Their approach extracts natural language descriptions of services from their metadata, also exploiting the fact that non-machine readable properties, such as the identifier name of a service, are often made up of very relevant human readable words. Their system comprises of different rules to extract such information from different properties of web services and from different web service description languages, such as OWL (Bechhofer et al. 2004) and WSMO (Roman et al. 2005). The usefulness of this approach is demonstrated by allowing non-technical users to discover relevant services using keywords-based queries. This related piece of work could be seen as a possible approach to allow human users to discover relevant services to link to their natural language instructions. Although it might not be possible to invoke these services automatically, they could be displayed as additional information to a user interested in achieving a task, who can then decide whether to invoke them or not.

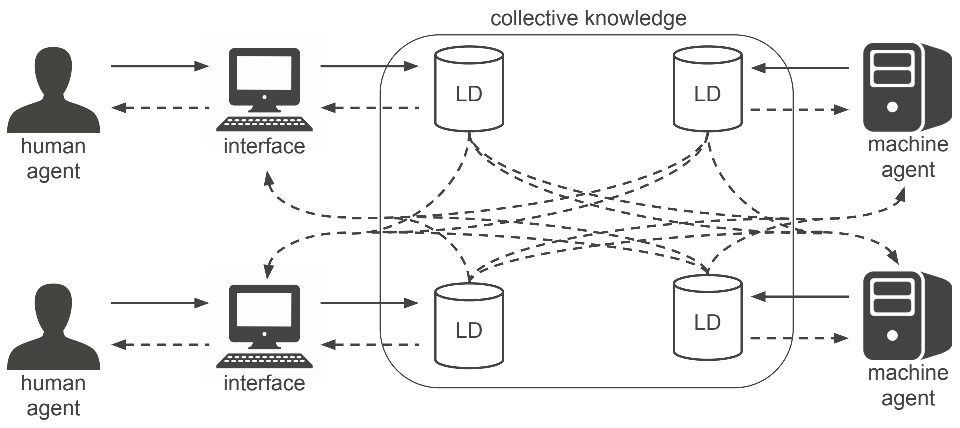


Figure 7.7: A multi-agent environment with no shared editable resources. Dashed and solid lines represent, respectively, the ability to read and write Linked Data resources.

7.4 Decentralised Know-How Execution

In the previous section I presented the prototype of a system capable of tracking the execution of a set of instructions and automatically complete its steps when possible. In this section I generalise this approach to automation to the case of multiple agents and decentralised resources. As its main contribution, this section presents a novel application of Linked Data as an indirect collaboration framework for humans and machines. I will discuss the main benefits and limitations of this framework and propose initial solutions to the problems of dynamic Linked Data discovery, of querying frequently-updated distributed datasets and of guaranteeing consistency in the case of concurrent updates by multiple agents. It should be noted that I will discuss this framework only at a conceptual level, as a concrete implementation of it has been not been developed and remains as a possible direction for future work. The contents of this section have been published as a workshop paper (Pareti 2016).

7.4.1 Problem Description

I propose a framework that allows a collaborative set of human and machine agents, who can publish and access web resources but cannot directly interact with each other, to *communicate* and *coordinate* their actions to collaboratively achieve tasks in the absence of a centralised system. To achieve decentralisation, no shared resource which multiple agents can edit is assumed to be available. As depicted in Figure 7.7, each agent is able to modify resources in its own repository while it is able to read the resources in all of the repositories of the other agents. In this context, *communication* refers to the process by which agents can propagate information (i.e. RDF triples) to the other agents by modifying the *collective knowledge*, namely the resources that all agents can access. *Coordination* instead refers to the ability to guarantee certain conditions across all datasets. For example, coordination might be required to ensure that no agent can declare its intention to execute a task which is already being executed by another agent.

This framework is intended to address *dynamic partially-automatable* problems. A partially-automatable problem requires some human intervention, as it cannot be entirely automated. At the same time, it cannot be optimally solved by humans alone, as some part of it could be automated. Dynamic problems, instead, are those that cannot be predicted in advance, making the development of dedicated software solutions impractical. To simplify the problem at hand, it is assumed that the agents involved already know and trust each other. Issues such as agent discovery, coalition formation and trust are considered outside the scope of this project.

7.4.2 Motivating Example

As a running example we take the following use-case. John is an employee of a company working on a project alongside with Ann. Every week, John and Ann write a report about the status of the project which then gets sent to the other members of the company. To do so, they agree on the following procedure:

Weekly report procedure:
- Step 1: John writes a draft of the report
- Step 2: The report is uploaded to an online repository
- Step 3: Ann corrects the report
- Step 4: The report is sent to all the colleagues

While this procedure can be completed manually, it can also be made more efficient with automation. For example, as soon as John finishes writing the draft of the report, a machine agent could upload it to the correct repository. Ann could then be automatically notified that she can start correcting the report as step 3 is ready to be executed. Step 4 might also be a good candidate for automation. Different colleagues might have different preferences on how to be contacted. For example, some might prefer to receive an email, others a message on an instant messaging system or a social networking platform. A machine agent could deal with these diverse preferences automatically, provided its given the necessary contact details.

7.4.3 Method

Lacking any central system, agents are the only components of the proposed framework. At a functional level of abstraction, all agents can be seen as entities capable of reading and writing Linked Data resources. Agents also have individual repositories where they can publish and modify Linked Data. More specifically, agents routinely perform three phases. During the first phase, agents "sense" the environment by accessing the Linked Data resources of the other agents. This phase is described in Section 7.4.3.1. During the second phase, agents can perform actions and decide what to communicate to the other agents by reasoning over the state of the environment and their own capabilities. Human agents perform this phase by interpreting the state of the environment using intuitive interfaces while machine agents use a logical formalism. This phase is described in Section 7.4.3.2. During the third phase, agents update their own repository with the information they intend to communicate to the other agents, such as the outcome of their actions. This phase is described in Section 7.4.3.3.

7.4.3.1 Knowledge Retrieval

While direct communication requires knowledge of the recipients of the messages, indirect communication requires a common environment that all agents can observe (Keil and Goldin 2006). An agent wanting to use direct communication must know how to transmit information to specific recipients. This would be a impractical in our scenario as agents might be humans or machines and might need to agree on different communication protocols. On the other hand, agents using indirect communication mechanisms only need to know how to store and retrieve information from the shared environment. Using indirect communication, agents can be oblivious to the characteristics or even the very existence of the other agents.

This framework adopts the same indirect communication mechanism presented in Section 7.3.4. In this framework, the shared environment that enables indirect communication is a set of Linked Data resources available online, here called *collective knowledge*. Therefore agents, to communicate, only need the ability to read and write Linked Data resources. It is important to notice that while all agents can access and modify the overall collective knowledge, they cannot modify all parts of it, but only the resources that they have direct control of.

Although agents do not need to have explicit knowledge of each other, they need to know where to access the resources that make up the collective knowledge. Implicitly,

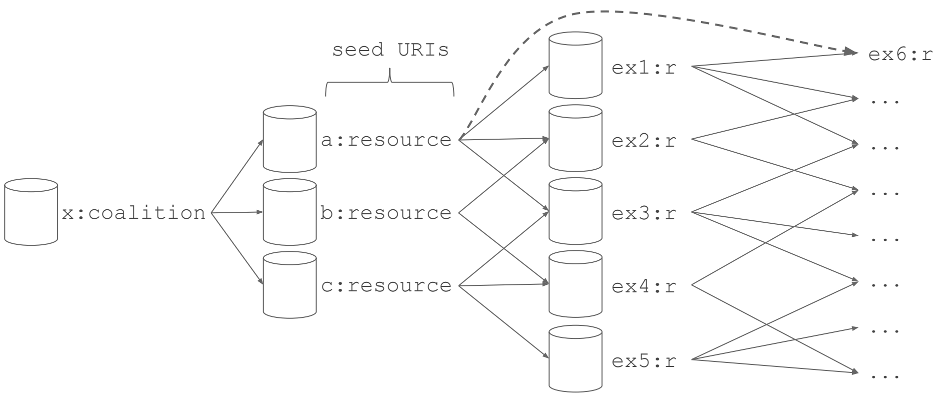


Figure 7.8: Dynamic Linked Data exploration can lead to the discovery of a large amount of resources. Adding the dashed link reduces the link distance from the seed URIs to `ex6:r` to 1.

this involves either knowing or being able to retrieve the addresses of the resources of the other agents. One possible approach is to agree on a URI for the collaboration. This URI should be dereferencable to an RDF resource containing the addresses of the other repositories in the collective knowledge. For example, we can imagine agent *a* joining coalition `x:coalition`. By dereferencing this URI, an agent might retrieve the following triples:

```
x:coalition prohow:includes a:resource, b:resource, c:resource .
```

The relation `prohow:includes` is possible extension to the PROHOW vocabulary to link the identifier of a coalition with the resources included in the collective knowledge of the coalition. These resources are here called *seed* URIs. If none of the seed URIs points to a resource editable by an agent *a*, then it can be inferred that agent *a* does not belong to this coalition, as it would not be possible for this agent to communicate information to the other agents. We will assume here that resource `a:resource` is editable by agent *a*. Agents do not necessarily know how many other agents are involved in the coalition, if any, as Linked Data resources might not belong to any agent, or multiple resources might belong to the same agent.

Agents wanting to agree on a coalition URI and on the list of seed URIs might have to resort to direct communication or to centralised systems, such as a matchmaking service. This process, however, is considered part of coalition formation and therefore outside of the scope of this work. In this section we assume that agents already have access to the list of seed URIs.

Having obtained the list of seed URIs, agents can now discover the collective knowledge of the coalition. This is done by dereferencing the URIs of the entities they are interested in, starting from the seed URIs. Agents should attempt to request machine-readable versions of the resources first, using content negotiation. The retrieved resources should then be correctly interpreted if encoded in one of the standard formats, such as Turtle, RDF/XML or RDFa. After retrieving all the data in the collective knowledge agents can proceed to reason about it by querying it. The process of querying distributed Linked Data resources by retrieving and storing them locally is a very common approach, and it has been identified by terms like *data warehousing*, *materialisation-based* approach (Umbrich et al. 2011) or data *consolidation* (Langegger 2008).

Dereferencing URIs allows agents to dynamically discover Linked Data resources at runtime. However, as depicted in Figure 7.8, the discovered resources might contain more URIs which lead to the discovery of even more resources, giving rise to a virtually unconstrained set of resources. As with web crawling, if no limit is imposed on the depth of the exploration, an excessively large number of resources could be discovered and included in the collective knowledge. Considering that agents need to frequently access all the resources in the collective knowledge, unconstrained exploration is impractical.

One possible solution to this problem is to limit the exploration of URIs only to one level from the seed URIs. In other words, only URIs directly retrieved beyond the seed URIs would be considered as potential resources in the collective knowledge. Agents discovering relevant resources by exploring beyond this limit can include them in the collective knowledge by adding their URIs in one of the seed URI resources. In principle, any RDF resource could be considered relevant by the agents and then added to the collective knowledge. For example, relevant resources could explain how to decompose a task that the agents need to perform. In Figure 7.8, for example, agent *a* can add `ex6:r` to the collective knowledge by adding a link to it in resource `a:resource`.

An alternative to dynamic Linked Data discovery is the addition of relevant data into a single repository. For example, agent *a* could copy the RDF data from resource `ex6:r` into its resource `a:resource`. Doing this would give more flexibility to agent *a* to decide which triples to include in the collective knowledge. By linking to resource `ex6:r` instead, agent *a* must commit to including all data from this resource in the collective knowledge. Moreover, copying data in the agent's repository could be considered a more stable approach in case the external resource has a high risk of becoming unavailable.

If the external resources are sufficiently stable and do not contain a significant amount of superfluous information, however, the dynamic Linked Data discovery approach seems more advantageous. Superfluous information is information contained in a resource that is not needed by the agents to collaborate. Linking to external resources and allowing their discovery avoids the problem of data duplication. Data duplication not only increases the amount of data hosting capability required by the agents, but it also makes updates more difficult. In fact, if the data duplication approach is followed, an update to repository `ex6:r` might take a long time to propagate to all the resources that contain its copy, or even not propagate at all. For these reasons, dynamic Linked Data discovery at runtime can be a practical solution if the discovered resources are stable and self-contained.

7.4.3.2 Knowledge Representation and Reasoning

In order to communicate meaningfully, a shared knowledge representation needs to be established. In this framework, this representation is the PROHOW vocabulary. For the reminder of this section I will assume that data is encoded as RDF and the agents share PROHOW as a common vocabulary to represent instructions and their executions. However, several approaches could be used in scenarios where these assumptions do not hold. For example, ontology matching techniques could be used to create semantic interoperability between different conceptualisations (Otero-Cerdeira et al. 2015). Also, systems such as the ones described in Chapter 5 can be used to translate non-RDF information into RDF.

One of the main challenges to enable communication between humans and machine agents is the representation of knowledge in a form which is both human and machine-understandable. It is therefore important to map such representation both to a logical formalism to allow machine reasoning, and to an intuitive representation, such as natural language, which can be understood by humans. Data modelled with the PROHOW vocabulary can be translated both into a natural language representation and into logical statements, as discussed in the previous section. Logical statements allow a machine to infer, for example, that a task is ready to be executed because if all of its requirements are complete, or that it has been accomplished if one of its methods has been completed.

While machines rely on logic, human reasoning and actions are mediated through *interfaces*, such as the one described in Section 7.3. In this context, an interface is defined as a software agent capable of translating Linked Data into a human readable representation and of translating human interactions into Linked Data.

7.4.3.3 Knowledge Update and Coordination

An agent wanting to communicate a set of triples to the other agents does so by publishing them in a repository which belongs to the collective knowledge. It will be the other agents' responsibility to retrieve these triples and to consider them in their reasoning process. This approach is straightforward in case the decision to upload certain triples is independent on the rest of the collective knowledge. However, coordination typically requires inferences based on the whole collective knowledge, and changes to the collective knowledge could potentially invalidate such inferences. For example, an agent's decision to complete a task might be overridden by the knowledge that the task has already been completed.

In RDF, changes to a knowledge base can be seen in terms of additions and deletions of triples. The problem of unexpected deletions of triples can be avoided by choosing a knowledge model that does not require triple deletion. The chosen model, PROHOW, is based on a monotonic increase of RDF triples and no facts need to be retracted in order to update a dataset. To avoid the problem of unexpected triple additions, agents should write potentially conflicting statements as "candidate" additions which are confirmed on a first-come first-served basis only after verifying that no other conflicting statement exists.

Candidate sets of triples can be written using RDF reification (Hayes 2004). Reification provides a unique identifier for each triple, which can be used to attach meta-information about the triple such as the timestamp at which the reified triple was created. Moreover, a reified triple does not entail the triple. This gives agents the flexibility to limit their reasoning only to confirmed facts or to consider also the candidate statements that other agents intend to assert.

Going back to the example presented in Section 7.4.2, we can imagine that both John and a machine agent a are capable of completing step `:t2`, "The report is uploaded to an online repository". In this scenario, it is desirable that only one of the agents accomplishes task `:t2`. This problem can be solved by coordination, and a simple way of implementing it is on a first-come-first-served basis as depicted in Figure 7.9. After retrieving the collective knowledge (timestamps t_1 to t_3) and discovering that task `:t2` needs to be completed, John publishes on his individual repository LD^1 the intention to complete it using timestamped reified statements at time t_4. However, agent a might have also published its intention to complete task `:t2` on repository LD^3

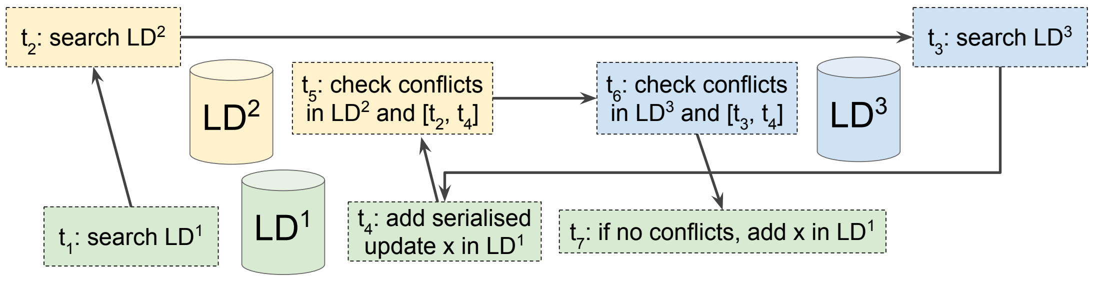

t$_2$: search LD^2
t$_3$: search LD^3
LD^2
t$_5$: check conflicts in LD^2 and [t$_2$, t$_4$]
t$_6$: check conflicts in LD^3 and [t$_3$, t$_4$]
LD^3
LD^1
t$_1$: search LD^1
t$_4$: add serialised update x in LD^1
t$_7$: if no conflicts, add x in LD^1

Figure 7.9: A simple coordination mechanism for distributed datasets.

at an earlier time t_x ($t_3 < t_x < t_4$).

Before considering his intention as final, John's interface will access the other agents' repositories to verify whether new conflicting statement have been added between timestamps t_5 and t_6, whether reified or not. When accessing resource LD^3 at time t_6, John's interface will discover the intention of agent a to accomplish task `:t2`. Since the other agent's statement has been created before John's one, the interface will warn John that task `:t2` is already in the process of being automated and he will be asked not to complete it. Agent a, instead, not having found any conflicting statements published before its own, will publish its intention to complete `:t2` in a non-reified format and proceed in accomplishing the task. The situation in which two conflicting statements are published at the same exact time can be resolved with different strategies. For example, both statements can be discarded and the agents can perform a second attempt at asserting them.

7.4.4 Complexity Analysis

The agents' behaviour can be seen as a loop where the three phases described in Section 7.4.3, namely knowledge retrieval, reasoning and knowledge update, are repeated with a certain frequency. I define frequency f as the number of times the iteration is repeated within a fixed time interval. During each iteration, agents need to retrieve the collective knowledge locally to reason over it. This process of retrieving data locally might need to be repeated a certain number of times k. The coordination algorithm described in Section 7.4.3.3 requires retrieving the data twice ($k = 2$) per iteration.

Within a set of agents A collaborating with each other, the average amount of data that an agent a needs to retrieve to access the collective knowledge is $d_A(|A|-1)$, where d_A is the average number of triples in the repositories of the agents in A. Therefore, within a fixed time interval, the average number of triples an agent has to retrieve from the web is $fkd_A(|A|-1)$. An agent collaborating with multiple sets of agents $\bar{A}$ only needs to retrieve the resources of each different agent once. Agents can merge data from each distinct agent it collaborates with ($\hat{A} = \bigcup_{A \in \bar{A}} A$) and use it to collaborate with all the other sets of agents in $\bar{A}$. Using this approach, the average number of triples an agent would need to retrieve is:

$$fkd_{\hat{A}}(|\hat{A}|-1) \tag{7.1}$$

From this formula it can be observed that the required web traffic increases linearly with the frequency at which the agent checks and computes updates f and the number of times the same resources need to be accessed per iteration k. In a concrete implementation of the system, both k and f can be considered constant. The remaining variables determining the complexity are the average number of triples in the knowledge bases of all the other collaborative agents ($d_{\hat{A}}$) and the absolute number of those agents $|\hat{A}|$. The web traffic generated by a single agent can then be considered as having complexity $O(d_{\hat{A}} * |\hat{A}|)$.

The space and computational complexity of the implementation of an agent depends to a large degree on the number of triples that needs to be stored and processed at each iteration, and it is therefore comparable to the web traffic complexity. The space and time complexity is also affected by other factors which depend on the specific implementation of an agent. These factors are the number and type of SPARQL queries that need to be executed, and the efficiency of the particular RDF storage and SPARQL engine used.

From this analysis we can observe that the complexity of each agent can be reduced by having agents collaborate with fewer other agents at a time and by having them reason over smaller datasets. It should be noted that, thanks to decentralisation, the complexity of the system does not depend on the total number of agents in the system nor on the total number of triples involved. This complexity can then be kept constant as the overall number of agents and knowledge in the system increases. This analysis suggests that this framework could be applied to a large number of lightweight agents, namely agents that interact with a small number of other agents and that do not publish a large volume of data.

This system incentivises agents to keep their number of collaborations within an acceptable limit because every additional collaboration the agent joins results in an increase in web trafficking complexity. To reduce this complexity even further, the web resources used in the collaboration should be small and free of any superfluous information. For example, an agent involved in multiple collaborations should consider using a different shared resource for each collaboration, so that agents involved in a particular collaboration are not forced to retrieve information about other collaborations too.

7.4.5 Discussion

In this section I proposed a novel framework for decentralised human-machine collaboration. To avoid the complexity of direct interactions between distributed human and machine agents, Linked Data is used as an indirect communication mechanism. The problem of coordination is therefore translated into a knowledge-sharing problem, where the only requirement for agent participation is the ability to retrieve and publish Linked Data. Linked Data used for collaboration is frequently updated and therefore agents need to constantly retrieve the most updated version. The necessity to regularly retrieve distributed resources imposes practical constraints on their size which results in Linked Data being divided into small and self-contained resources. This framework can be seen as a promising application scenario for URI dereferencing to discover Linked Data resources at runtime thanks to the small size of those resources and the necessity to retrieve them frequently.

The type of indirect communication used in this framework resembles Blackboard Systems (Corkill 1991). In BS, multiple agents collaborate to compute the solution to a problem by modifying a shared resource (the blackboard). Fensel et al. (2011) describe another related communication mechanism which originates from Blackboard Systems: Triple Space Computing (TSC). 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*.

The proposed framework differs from Blackboard Systems for two main reasons. First of all, in the approach I have proposed the shared environment, or collective knowledge, is fragmented into multiple resources which cannot be accessed simultaneously. A perfectly updated view of the collective knowledge is therefore impossible, as during the time required by an agent to retrieve a single resource, all the other resources might have potentially changed. Secondly, agents can only modify their own resources and no resource can be modified by more than one agent. This means that agents cannot agree on a single resource to be used for coordination. The indirect collaboration mechanism of the proposed framework can also be seen as a form of stigmergy, a term introduced in Section 7.3.4. In the context of multi-agent systems the term stigmergy traditionally refers to complex collaborations emerging from simple agents by indirect interactions mediated through an environment. In the case of rational agents, the concept of *cognitive stigmergy* has been proposed by Ricci et al. (2007).

Given the frequent updates to the collective knowledge of all the agents, I have chosen to query the distributed Linked Data resources using a materialisation-based approach. If no assumptions can be made on which resources are likely to be updated, and how frequently, this approach guarantees that any update to the collective knowledge will propagate to all agents after at most one iteration. Under different assumptions, other distributed query strategies could be more efficient. One class of approaches assumes that query processing capabilities, such as SPARQL endpoints, are available remotely (Quilitz and Leser 2008). Under this assumption, queries could be split into several sub-queries that will be evaluated remotely against the remote sources. The other class of approaches, to which materialisation-based approaches belong to, does not make this assumption and requires remote resources to be retrieved locally before they can be queried. This process can be optimised by creating local indexes of the remote resources based on their schema (Stuckenschmidt et al. 2005), based on the URIs they contain, or based on both of those properties (Umbrich et al. 2011). This information is used to decide which resources to access when computing a query, potentially avoiding the retrieval of irrelevant resources.

The proposed materialisation-based approach is designed to deal with the high frequency of updates and the need to compute exact answers. I proposed an algorithm to coordinate the potentially concurrent updates of distributed resources by multiple agents. The complexity analysis of this system that I conducted shows that this framework can scale to a large number of agents as long as the resources shared by the agents are kept small in size, and agents collaborate only with a limited number of other agents at a time.

7.5 Conclusion

In this chapter I have provided examples of the possible uses of the PROHOW model and of alternative formalisation of instructions. In particular, I have assessed the suitability of the two main applications that this model is intended to support, namely know-how discovery and execution. In Section 7.1 I presented HowLinks, a visualisation tool for PROHOW data designed to facilitate the exploration of human know-how and of related factual knowledge. The advantages brought by the PROHOW model were discussed with respect to "exploration goals", namely queries that a user might want to answer by navigating a set of resources. These advantages were exemplified by five exploration goals that HowLinks allows users to achieve directly, but that would require significant user efforts using existing instructional websites.

In Section 7.3 I presented a framework to enable humans and machines to collaborate in the execution of a task. This collaboration is enabled by sharing a PROHOW dataset that is both human and machine-understandable. The human interpretation of this dataset is mediated through a visual interface that translates PROHOW data into standard step-by-step lists of instructions and their execution into an interactive to-do checklist. The machine interpretation of this dataset follows the logical inference rules of the PROHOW model to allow the system to determine when its functionalities are needed and later to update the state of the execution. An implementation of this system was described, along with experiments that support the hypothesis that the majority of non-programmer users can define PROHOW instructions at a level of abstraction which is compatible with machine functionalities. As discussed in Section 7.3.7, a novelty of this application compared to previous research is the ability of users to be in charge of defining the details of the human-machine collaboration they are interested in.

The possibility of decentralising these applications has been investigated in Sections 7.2 and 7.4. These sections deal with a more complex but realistic scenario where the PROHOW representation of multiple sets of instructions is not available in a single dataset, but it is decentralised as distributed Linked Data resources managed by different agents. I have presented a concrete implementation of a tool that can explore these resources dynamically. Moreover, I proposed a conceptual framework that could allow multiple human and machine agents to collaborate to the execution of tasks by sharing these Linked Data resources.

In conclusion, the two main applications discussed in this chapter demonstrate how the PROHOW formalisation of instructions can be said to be more understandable by machines than their original HTML representation. This increase in machine understanding was measured as an increase in automation of realistic user goals. This chapter also demonstrates how the information formalised by the PROHOW model not only supports typical factual knowledge applications such as improved exploration and semantic queries. It also supports applications which are specific to procedural knowledge, such as the automated execution of instructions.

These applications, along with related work on computational uses of instructions, give an overview of the benefits of formalising human know-how. This is a research area where much progress can still be made, but that nevertheless already offers demonstrable benefits.

Chapter 8

Conclusion

An increasingly larger portion of human knowledge is being uploaded on the web, which is now, for many web users, a major source of information in everyday life. This knowledge is available for everyone to access, including by machines. Web resources hold a great potential for computational systems, which can process them in large amounts to extract reusable patterns and to find answers to complex queries.

To date, these resources have been mostly analysed from the point of view of *factual knowledge*. In other words, with the objective of learning facts about the world, such as the population of cities or the results of sport competitions. However an important type of knowledge, namely *procedural knowledge*, or *know-how*, is still largely inaccessible by machines despite figuring prominently on the web. In this thesis I focus on human textual instructions, a common form of procedural knowledge on the web. The main problem that I addressed are the limitations caused by the fact that human instructions on the web are not machine-understandable. Such limitations are, for example, an inability to answer complex queries about instructions, and the lack of automated support that can be offered to users interested in following instructions.

I have broken this problem down into three sub-problems, namely (1) the formalisation of know-how in a machine-understandable way (know-how representation), (2) the acquisition of such formalisation from existing resources (know-how acquisition) and (3) the use of this formalisation to support different applications (know-how applications). In Chapter 1 I discussed how these three problems relate to each other and why they should not be researched in isolation. The know-how acquisition and know-how applications problems, in fact, result in conflicting requirements on how to address the know-how representation problem. More specifically, the choice of a very abstract formalisation of instructions would ease the acquisition problem but its lack of details would prevent its effective use in many applications. Conversely, a very rich formalisation would be useful in many applications but would be difficult to acquire automatically.

In Chapter 2 I reviewed the current state of the art with respect to this problem. This highlighted the rich amount of literature concerned with procedural knowledge coming from different areas such as Automated Planning, Web Services and Business Workflows. The more specific topic of instructional know-how, however, has not been investigated in depth in the literature. Of the projects concerned with instructional know-how, none offers a systematic study of all of the three sub-problems mentioned in this thesis.

8.1 Main Contributions

In this thesis I have presented and evaluated the first method that addresses simultaneously all of the three sub-problems of making know-how more understandable by machines. This method is centred around the mathematical model of human instructions presented in Chapter 4. I designed this model, called PROHOW to be compatible both with human instructions and with existing process formalisms. To achieve this, several instructional websites were analysed to extract a number of common concepts, such as *steps*, *methods* and *requirements*, which were then integrated in the model. At the same time, I have shown how the concepts used in this model can be mapped to standard concepts used in process formalisms, such as *inputs*, *outputs* and *activity decompositions*. As an example of this I have provided a concrete mapping between PROHOW and PDDL (Mcdermott et al. 1998), a language to define formal Automated Planning problems.

The development of the PROHOW model followed the NeOn method described by Suárez-Figueroa et al. (2012) and it was guided by a number of *competency questions*, namely the type of questions that the knowledge base should be able to answer. These questions were formulated with respect to two main scenarios: namely know-how exploration and know-how execution. In the know-how exploration scenario, the knowledge base should be able to support a user in finding more information about how a certain task can be achieved. This requires the ability to answer questions such as: "what are the steps of task t?".

In the know-how execution scenario, the knowledge base should be able to support a user in accomplishing a set of instructions. To do so, I have formalised an execution model of instructions that follows the intuition behind check-lists. Similarly to check-lists, the execution of a complex task is decomposed into smaller tasks, or steps, which are "checked off" when complete. The particular properties of the set of instructions dictate constraints on this execution, for example mandating a specific ordering of the steps, and specifying which specific set of steps can be considered sufficient to achieve a more complex task. In order to support this type of execution, the knowledge base should be able to answer competency questions such as "which tasks have already been checked-off?" and "which tasks should I complete next to achieve a certain goal?".

I have demonstrated how these competency questions can be answered by means of Automated Planning following the PDDL translation of the model. At the same time, I have demonstrated how these competency questions can be answered by the PROHOW vocabulary: a concrete instantiation of this model in RDF. The choice of using a Semantic Web technology such as RDF allows this vocabulary to be easily published, accessed and integrated on the web. The experiments conducted in these thesis were based on this concrete instantiation of the PROHOW model.

Having defined a specific model to represent instructional knowledge, two problems remained to address, namely know-how acquisition and know-how applications. In Chapter 5 I have provided a method to acquire PROHOW representations of instructions from a specific class of instructional resources, namely *semi-structured resources*. This method exploited the regular template of certain instructional websites, such as a standard formatting of steps and requirements, to acquire a complete and correct formalisation of instructions in PROHOW. I have tested this method on two instructional websites, namely wikiHow and Snapguide, generating a large-scale dataset of formalised instructions.

In Chapter 6 I have described a know-how interlinking method based on Natural Language Processing and Machine Learning to automatically integrate PROHOW instructions with each other and with DBpedia. This method discovers links from steps of instructions and other instructions that explain how to perform them. For example, the step "Prepare a job application" would be linked to the set of instructions on "How to prepare a job application". This method also discovers links from the objects found in a set of instructions, such as "3 eggs", and related DBpedia resources, such as the DBpedia resource for "Eggs".

To evaluate the quality of this know-how interlinking method I have applied it to the dataset acquired from the wikiHow and Snapguide. I have compared the results of this experiment with the equivalent links present in wikiHow instructions created by its community of users. The results of this evaluation show that the proposed interlinking method has achieved higher precision and recall. The dataset acquired from wikiHow and Snapguide and enriched with these links has been published online.[1]

Lastly, in Chapter 7 I have demonstrated how this formalisation can support two different applications, namely know-how exploration and know-how execution, in ways that were not possible without a machine-understandable representation of instructions. To do this, I have developed two know-how exploration tools. The first one, HowLinks, provides an integrated visualisation of PROHOW instructions in a single tree structure that users can expand to explore know-how in novel ways. A subset of the features of HowLinks is also available through the KnowHow4j tool that I have developed. Unlike HowLinks, however, KnowHow4j is a lightweight Javascript application that can be run in any modern web browser and that can dynamically explore PROHOW data from decentralised sources. KnowHow4j can also be used by users to produce sets of human-readable instructions as HTML pages with embedded machine-readable PROHOW data.

The possibility to use the PROHOW formalisation to support know-how execution was also presented in Chapter 7. To do this, I have developed a system that generates PROHOW check-lists from sets of instructions. These check-lists are visualised by users as an interactive HTML document. Users can "check" boxes to declare tasks complete and optionally ask the system to automate some of the steps. I have tested the usability of this system by users with no programming experience. Human participation has been acquired through the Crowdflower crowdsourcing platform. In these experiments, the majority of the workers managed to formalise a specific problem as a set of instructions and link it to a set of machine functionalities. These functionalities could then be used to automate certain steps of the instructions.

I have investigated the possibility to extend this type of execution system into a decentralised scenario where multiple human and machine agents can collaborate to the execution of a task. The result of this investigation is a conceptual framework that could enable the collaboration between humans and machines. In this framework, direct communication between the agents is not required, and the only requirement for participation is the possibility to publish and share PROHOW data.

Overall, the work presented in this thesis addressed a novel combination of research problems. However, this novelty resulted in a number of challenges, the most significant being the lack of a previous baseline to compare to, and the lack of a well

[1]`https://w3id.org/knowhow/dataset` (accessed on 17/10/2017)

established methodology for evaluation. As discussed in Chapter 3, I addressed these challenges by creating a new methodology that incorporates evaluation methods from different fields.

More specifically, I have evaluated my approach to the know-how representation problem by analysing the chosen knowledge formalisms with respect to its expressiveness and logical properties. This has been done using methods such as competency questions, which are common in the research areas about ontologies and semantic technologies. My approach to the know-how acquisition problem, instead, has been evaluated using experimental methods commonly found in areas such as information extraction and data integration. These methods involve processing large volumes of real-world data to compute the precision and recall metrics of the acquired knowledge. Finally, I have evaluated my approach to enable know-how applications by the creation of application prototypes. Those prototypes demonstrate novel uses of procedural knowledge and provide insights on the challenges that are involved in deploying them.

8.2 Discussion

This body of work represents an initial solution to the three problems addressed in this thesis. The PROHOW model I developed answers the know-how representation question about how to formalise instructional knowledge in a machine-understandable form. Using this model, the know-how acquisition and know-how applications problems were shown to be solvable. To do so, I demonstrated concrete methods that answer both (1) the question on how to acquire knowledge in this formalisation from existing resources, and (2) the question on how to use this formalisation of instructions to support novel applications. By addressing all three sub-problems at once, this thesis provides the first complete approach that can be used to make web-instructions more machine-understandable.

The main problem addressed in this thesis, however, cannot be considered completely solved. The generation of know-how resources on the web that can be understood by machines is a grand goal, and the work in this thesis only represents an initial step in this direction. Further improvement is possible in any of the three sub-problems, for example by generating semantically richer representations of instructions, which can be acquired from a wider range of resources and used to support more advanced applications.

One of the main limitations of my work, for example, is that instructional knowledge can be acquired reliably only from semi-structured web resources. Moreover, the experiments I conducted focus on two semi-structured websites, and decisive evidence of the applicability of my approach to other similar websites has not been gathered. Methods to generate PROHOW representations from instructional knowledge in other forms would extend the practical applicability of my work. Given the large amount of existing semi-structured resources, however, I argue that this is not a severe limitation.

Although much progress can still be made, the method I proposed already provides tangible benefits. In particular, the interlinking method presented in Chapter 6 produces links between sets of instructions which are of higher quality than the ones being used on wikiHow, and which can be generated automatically. Therefore, these links can be said to be of sufficient quality to be useful to human users, and their addition on the wikiHow and Snapguide websites could directly benefit their users.

On websites like wikiHow, the components of a set of instruction can be efficiently linked to up to one external resource by adding an HTML link in the textual representation of the components. By enriching these instructions with my methodology, web pages might require certain visualisation changes to accommodate for a larger number of links. The HowLinks visualisation tool I presented in Chapter 7 represents one way to do this. Moreover, it demonstrates how formalised instructional knowledge, when de-coupled from its visual environment, such as an HTML page, can be easily visualised through different interfaces.

The benefits that would result from more machine-understandable know-how are not just theoretical, but they are also attainable in practice. The experiments described in Chapters 5 and 6 demonstrate the scalability of my know-how extraction and interlinking methods when applied to large datasets containing hundreds of thousands sets of instructions. In particular, efficient strategies have been developed to significantly reduce the complexity of the interlinking method which, in its naive form, is exponential.

Large amounts of formalised instructions can also be stored in a scalable way by making use of decentralisation. In fact, these formalisations can then be efficiently and seamlessly stored in the same web pages that contain their human-understandable representation without the need of a central system. The KnowHow4j tool that I developed is a demonstration of this, as it allows the generation of human readable HTML pages with embedded machine readable PROHOW formalisations. These formalisations, which are embedded as RDFa metadata, are completely transparent to the user.

The decentralised systems I described in Chapter 7 demonstrate some of the possible uses of PROHOW formalisations even when such formalisations are decentralised. In this thesis, human and machine-understandable know-how can be created, hosted, and used by any user independently, with the aid of simple tools like KnowHow4j, which can run as scripts in any modern web browser. Arguably, the barrier of entry to use this technology is not much higher than the one to create and use traditional unstructured sets of instructions. Overall, I have demonstrated in a real-world scenario that a single formalism can be used to represent instructional knowledge in a way that is abstract enough to be easily acquirable from existing instructions, and rich enough to enable the development of novel applications.

8.3 Future Work

The work in this thesis presents several opportunities for future research. For example, the execution of PROHOW instructions could benefit from the integration with existing systems that allow non-programmer users to define simple executable commands. The applicability of this work, instead, could be extended to unstructured resources by integrating existing know-how acquisition systems with the PROHOW model. This integration could also be useful to compare these systems which, as it was discussed in Section 2.3.1, are not directly comparable due to the semantic heterogeneity of their formalisms. In this scenario, PROHOW could act as a shared semantic model of instructions.

The functional links occurring across instructional data, such as task decomposition links, can be studied as a generic and domain-independent class of links that current systems, focussed on discovering links based on similarity, cannot efficiently deal with. This could be investigated from the perspective of the Data Integration and Link Discovery fields. Alternatively, this link discovery problem could be used as a benchmark for the research of Machine Learning systems targeting extreme class imbalance.

Interlinked know-how can be seen as a type of semantic data, or Linked Data. Currently, semantic data on the web is, for the most part, a collection of datasets that can be retrieved and integrated with relative ease. This process is generally performed locally and web data is rarely used *on the web*. On the contrary, formalised know-how can be seen as a promising scenario not only for decentralising web data, but also to decentralise its use. My initial experiments suggest that currently under-used technologies, such as dynamic Linked Data discovery, and Linked Data processing by web browsers, which are still impractical in many applications, could play a major role in the domain of formalised human know-how.

In conclusion, the work presented in this thesis can be seen as the first comprehensive exploration of the opportunities and research challenges of pursuing machine-understandable instructional knowledge, which remains a promising direction for future work across a diverse range of fields.

Appendix A

The PROHOW Vocabulary

Listing A.1 shows a complete and concrete representation of the PROHOW vocabulary as an OWL file serialised in the Turtle format. This representation was produced by the Protégé ontology editing tool.[1]

Listing A.1: The PROHOW vocabulary represented in the Turtle serialisation of OWL.

```
@prefix : <http://w3id.org/prohow#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix prohow: <http://w3id.org/prohow#> .
@base <http://w3id.org/prohow#> .

<http://w3id.org/prohow#> rdf:type owl:Ontology ;
    owl:versionIRI :v1 ;
    <http://www.w3.org/1999/xhtml/vocab#license> <http://
        creativecommons.org/licenses/by/4.0/> .

#################################################################
#    Annotation properties
#################################################################

###  http://w3id.org/prohow#has_result
prohow:has_result rdfs:comment "The relation between an execution
   and its result (e.g. success or failure)."@en ;
```

[1]http://protege.stanford.edu (accessed on 17/10/2017)

```
                  rdfs:label "Has Result"@en ;
                  rdf:type owl:AnnotationProperty ;
                  rdfs:domain prohow:execution .



###  http://w3id.org/prohow#has_value
prohow:has_value rdfs:comment "The relation between an execution and
    its associated value."@en ;
                 rdfs:label "Has Value"@en ;
                 rdf:type owl:AnnotationProperty ;
                 rdfs:domain prohow:execution .



#################################################################
#    Object Properties
#################################################################

###  http://w3id.org/prohow#binds_to
prohow:binds_to rdf:type owl:ObjectProperty ;
                rdfs:domain prohow:task ;
                rdfs:range prohow:task ;
                rdfs:comment "The relation between two tasks that
                   can be interchanged."@en ;
                rdfs:label "Binds To"@en .



###  http://w3id.org/prohow#has_constant
prohow:has_constant rdf:type owl:ObjectProperty ;
                    rdfs:domain prohow:task ;
                    rdfs:range prohow:task ;
                    rdfs:comment "The relation between a task and a
                       task with a pre-defined value."@en ;
                    rdfs:label "Has Constant"@en .



###  http://w3id.org/prohow#has_environment
prohow:has_environment rdf:type owl:ObjectProperty ;
                       rdfs:domain prohow:execution ;
                       rdfs:range prohow:environment ;
                       rdfs:comment "The relation between an
                          execution of a task, and its environment
                          that defines its scope."@en ;
                       rdfs:label "Has Environment"@en .


###  http://w3id.org/prohow#has_goal
prohow:has_goal rdf:type owl:ObjectProperty ;
                rdfs:domain prohow:task ;
                rdfs:range prohow:environment ;
                rdfs:comment "The relation between an environment
                   and its goal."@en ;
                rdfs:label "Has Goal"@en .


###  http://w3id.org/prohow#has_method
prohow:has_method rdf:type owl:ObjectProperty ;
                  rdfs:domain prohow:task ;
                  rdfs:range prohow:task ;
                  rdfs:comment "The relation between a task and one
                     of its methods. If at least one method of a
                     task is achieved, then the task can be
                     considered achieved as well."@en ;
                  rdfs:label "Has Method"@en .


###  http://w3id.org/prohow#has_step
prohow:has_step rdf:type owl:ObjectProperty ;
                rdfs:domain prohow:task ;
                rdfs:range prohow:task ;
                rdfs:comment "The relation between a task and one of
                    its steps. If all the steps of a task are
                   achieved, then the task can be considered
                   achieved as well."@en ;
                rdfs:label "Has Step"@en .


###  http://w3id.org/prohow#has_task
prohow:has_task rdf:type owl:ObjectProperty ;
                rdfs:domain prohow:execution ;
                rdfs:range prohow:task ;
                rdfs:comment "The relation between an execution, and
                    the task that it was intended to accomplish."@en
                    ;
                rdfs:label "Has Task"@en .
```

```
###  http://w3id.org/prohow#requires
prohow:requires rdf:type owl:ObjectProperty ;
                rdfs:domain prohow:task ;
                rdfs:range prohow:task ;
                rdfs:comment "The relation between a task and one of
                     its requirements. If all the requirements of a
                    task are satisfied, then the task is ready to be
                    executed."@en ;
                rdfs:label "Requires"@en .


###  http://w3id.org/prohow#requires_one
prohow:requires_one rdf:type owl:ObjectProperty ;
                    rdfs:domain prohow:task ;
                    rdfs:range prohow:task ;
                    rdfs:comment "The relation between a task and a
                       sufficient requirement. If at least one
                       sufficient requirement of a task is satisfied
                       , then the task is ready to be executed."@en
                       ;
                    rdfs:label "Requires One"@en .


###  http://w3id.org/prohow#sub_environment_of
prohow:sub_environment_of rdf:type owl:ObjectProperty ;
                          rdfs:domain prohow:environment ;
                          rdfs:range prohow:environment ;
                          rdfs:comment "The relation between an
                             environment and the environment that
                             generated it."@en ;
                          rdfs:label "Sub Environment Of"@en .


#################################################################
#    Classes
#################################################################

###  http://w3id.org/prohow#environment
prohow:environment rdf:type owl:Class ;
```

```
                   rdfs:comment "The scope of one particular
                      intention at achieving a goal."@en ;
                   rdfs:label "Environment"@en .



###  http://w3id.org/prohow#execution
prohow:execution rdf:type owl:Class ;
                 rdfs:comment "A particular attempt at achieving a
                    task."@en ;
                 rdfs:label "Execution"@en .



###  http://w3id.org/prohow#task
prohow:task rdf:type owl:Class ;
            rdfs:comment "Something that can be accomplished by
               doing some action."@en ;
            rdfs:label "Task"@en .



#################################################################
#    Annotations
#################################################################

prohow:complete rdfs:comment "The concept of a succesfully
   accomplished task."@en ;
                rdfs:label "Complete"@en .
```

Appendix B

PROHOW to PDDL Conversion

This appendix presents the translation of the PROHOW model into the PDDL language. The code presented in this appendix is also available on GitHub.[1] Listing B.1 contains the axioms of the PROHOW model encoded as a PDDL domain description. The actions `accomplish_task` and `instantiate_sub_env` are not part of the basic PROHOW model, but they allow a planner to simulate the execution of a task, and the decomposition of a task into a separate sub-environment. If this extension is not needed, the actions `accomplish_task` and `instantiate_sub_env` should be deleted from Listing B.1. The `accomplish_task` action also includes the "primitive tasks" extension. Under this extension, the planner will only be able to directly achieve tasks which are defined as *primitive*. If this is extension is not needed, the line `(primitive ?t)` should be removed from action `accomplish_task`. Without this extension, however, the planner would not consider decomposing a super-task into sub-tasks as it can accomplish the super-task directly.

Listing B.1: The axioms of the PROHOW model as a PDDL domain description.

```
(define (domain prohow)
  (:requirements :strips :existential-preconditions :typing :
     equality)
  (:types task execution environment success finished primitive
     value)
  (:predicates
  (task ?x) (execution ?x) (environment ?x) (success ?x) (finished ?
     x) (primitive ?x) (active ?x)
  (requires ?x ?y) (requires_one ?y ?x) (has_step ?y ?x) (has_method
      ?y ?x) (has_task ?y ?x) (has_goal ?y ?x) (binds ?y ?x) (
```

[1]`https://github.com/paolo7/prohow-pddl` (accessed on 17/10/2017)

```
    has_constant ?y ?x) (has_value ?y ?x)  (has_env ?y ?x) (sub_env
     ?y ?x) (complete ?y ?x) (ready ?y ?x)  (related ?y ?x) (
    infer_active ?x)
(value ?y ?x ?z)
)

(:action accomplish_task
  :parameters (?t - task ?e - environment ?i - execution)
  :precondition (and
      (primitive ?t) ; remove to disable the primitive task
         extension
      (ready ?t ?e)
      (not
        (exists (?x)
          (has_task ?i ?x)))
      (not
        (exists (?x)
          (has_env ?i ?x))))
  :effect (and
    (success ?i)
    (has_env ?i ?e)
    (has_task ?i ?t)))

(:action instantiate_sub_env
  :parameters (?e - environment ?a - environment ?t - task ?m -
     task)
  :precondition(and
    (has_method ?t ?m)
    (not (= ?e ?a))
    (not
      (exists (?x)
        (sub_env ?e ?x)))
    (not
      (exists (?x)
        (has_goal ?e ?x))))
  :effect (and
      (sub_env ?e ?a)
      (has_goal ?e ?t)))

(:action infer_active
  :parameters (?e - environment)
  :precondition (and
    (exists (?g) (has_goal ?e ?g))
    (or (not (exists (?a) (sub_env ?e ?a)))
      (exists (?a ?x)
        (and
          (sub_env ?e ?a)
          (has_goal ?e ?x)
          (ready ?x ?a)
          (active ?a)))))
  :effect (active ?e))

(:action infer_complete
  :parameters (?y - task ?e - environment)
  :precondition (exists (?i)
    (and
      (has_env ?i ?e)
      (has_task ?i ?y)
      (success ?i)))
  :effect (complete ?y ?e))

(:action infer_finished
  :parameters (?e - environment)
  :precondition (exists (?g)
    (and
      (has_goal ?e ?g)
      (complete ?g ?e)))
  :effect (finished ?e))


(:action infer_complete_by_steps
  :parameters (?y - task ?e - environment)
  :precondition
  (and
    (exists (?x)
      (has_step ?y ?x))
    (not
      (exists (?x)
        (and
          (has_step ?y ?x)
          (not
              (complete ?x ?e))))))
  :effect (complete ?y ?e))
```

```
(:action infer_ready
  :parameters (?y - task ?e - environment)
  :precondition (and
    (active ?e)
    (or
      (not
        (exists (?x)
          (has_step ?x ?y)))
      (exists (?x)
        (and
          (has_step ?x ?y)
          (ready ?x ?e))))
    (or
      (not
        (exists (?x)
          (or
            (requires ?y ?x)
            (requires_one ?y ?x))))
      (and
        (exists (?x) (requires ?y ?x))
        (not
          (exists (?x)
            (and
              (requires ?y ?x)
              (not (complete ?x ?e))))))
      (exists (?x)
          (and
            (requires_one ?y ?x)
            (complete ?x ?e)))))
  :effect (ready ?y ?e))

(:action infer_complete_by_method
  :parameters (?y - task ?e - environment)
  :precondition (exists (?x)
    (and
      (has_goal ?e ?y)
      (has_method ?y ?x)
      (complete ?x ?e)))
  :effect (complete ?y ?e))

(:action infer_complete_by_sub_env
  :parameters (?y - task ?e - environment)
```

```
    :precondition (and
      (ready ?y ?e)
      (exists (?a)
        (and
          (complete ?y ?a)
          (sub_env ?a ?e)
          (has_goal ?a ?y))))
    :effect (complete ?y ?e))

  (:action infer_related
    :parameters (?a - environment ?e - environment)
    :precondition (or
      (= ?a ?e)
      (sub_env ?a ?e)
      (sub_env ?e ?a)
      (exists (?x)
        (and
          (related ?a ?x)
          (related ?x ?e))))
    :effect (related ?a ?e))

  (:action infer_value
    :parameters (?z - value ?y - task ?e - environment)
    :precondition (exists (?i)
      (and
        (has_env ?i ?e)
        (has_task ?i ?y)
        (success ?i)
        (has_value ?i ?z)))
    :effect (value ?z ?y ?e))

  (:action infer_complete_by_binding
    :parameters (?x - task ?e - environment)
    :precondition (and
      (ready ?x ?e)
      (exists (?y ?a)
        (and
          (binds ?x ?y)
          (complete ?y ?a)
          (related ?a ?e))))
    :effect (complete ?x ?e))
  (:action infer_value_from_binding
    :parameters (?z - value ?x - task ?e - environment)
    :precondition (and
      (ready ?x ?e)
      (exists (?y ?a)
        (and
          (binds ?x ?y)
          (value ?z ?y ?a)
          (related ?a ?e))))
    :effect (value ?z ?x ?e))

  (:action infer_complete_from_constant
    :parameters (?x - task ?e - environment)
    :precondition (exists (?i ?y)
      (and
        (has_env ?i ?e)
        (has_task ?i ?y)
        (has_constant ?y ?x)))
    :effect (and
      (complete ?x ?e)
      (value ?x ?x ?e)))
)
```

B.1 Sample Problems

A sample planning problem for the PDDL domain definition of the PROHOW model is presented in listing B.2. Listing B.4 details a plan generated to solve this problem by the Fast-Downward planner[2] using the “LAMA 2011 configuration”. Listing B.3 details a plan generated using the same configuration but without the primitive tasks extension.

Listing B.2: A PROHOW check-list defined as a PDDL problem.

```
(define (problem sample)
  (:domain prohow)
  (:objects
   t a b r x - task
   e ee - environment
   ex1 ex2 ex3 - execution)
  (:init
```

[2]http://www.fast-downward.org/ (accessed on 17/10/2017)

```
    (has_goal e t)
    (requires t r)
    (has_step t a) (has_step t b)
    (has_method b x)
    (primitive a) (primitive x) (primitive r))
  (:goal (finished e)))
```

Listing B.3: A plan generated by the Fast-Downward planner to solve the problem defined in Listing B.2 under the PDDL domain description of the PROHOW model without the primitive task extension.

```
(infer_ready_by_requirements r e)
(accomplish_task r e ex3)
(infer_complete r e)
(infer_ready_by_requirements t e)
(accomplish_task t e ex2)
(infer_complete t e)
(infer_finished e)
```

Listing B.4: A plan generated by the Fast-Downward planner to solve the problem defined in Listing B.2 under the PDDL domain description of the PROHOW model with the primitive task extension.

```
(infer_ready_by_requirements r e)
(infer_ready_by_requirements x ee)
(instantiate_sub_env ee e b x)
(accomplish_task r e ex)
(infer_complete r e)
(accomplish_task x ee ex2)
(infer_ready_by_requirements t e)
(infer_ready_by_requirements a e)
(infer_ready_by_requirements b e)
(accomplish_task a e ex3)
(infer_complete x ee)
(infer_complete_by_method b ee)
(infer_complete_by_sub_env b e)
(infer_complete a e)
(infer_ready_by_requirements x e)
(infer_complete_by_steps t e)
(infer_finished e)
```

The problem defined in Listing B.5 demonstrates the use of bindings. This problem is similar to the previous one, with the exception of a new requirement $r2$ of task x.

Requirement *r*2 is not in the set of primitive tasks. However, it has a binding with task *r*. The solution found to this problem and shown in Listing B.6 demonstrates how the planner manages to reach the goal by correctly inferring that task *r*2 is complete after completing task *r*.

Listing B.5: A PROHOW check-list containing bindings defined as a PDDL problem.

```
(define (problem sample)
  (:domain prohow)
  (:objects
   t a b r x r2 - task
   e ee - environment
   ex1 ex2 ex3 - execution)
  (:init
    (has_goal e t)
    (requires t r)
    (has_step t a )
    (has_step t b )
    (has_method b x)
    (requires x r2)
    (binds r2 r)
    (primitive x) (primitive r) (primitive a) )
  (:goal (finished e) ))
```

Listing B.6: A plan generated by the Fast-Downward planner to solve the problem defined in Listing B.5 under the PDDL domain description of the PROHOW model with the primitive task extension.

```
(infer_ready_by_requirements r e)
(infer_ready_by_requirements r ee)
(instantiate_sub_env ee e b x)
(accomplish_task r e ex1)
(infer_ready_by_requirements r2 ee)
(infer_related e ee)
(infer_complete r e)
(infer_complete_by_binding r2 ee)
(infer_ready_by_requirements x ee)
(infer_ready_by_requirements t e)
(infer_ready_by_requirements a e)
(accomplish_task x ee ex3)
(infer_ready_by_requirements b e)
(accomplish_task a e ex2)
(infer_complete a e)
```

```
(infer_complete x ee)
(infer_complete_by_method b ee)
(infer_complete_by_sub_env b e)
(infer_related ee ee)
(infer_complete_by_steps t e)
(infer_finished e)
```

To demonstrate the use of constants, a new action should be added to the model. This action, defined in Listing B.7, is called `instantiate_execution`(t,i,e) and it allows to instantiate a new execution i of task t in environment e. The problem defined in Listing B.8 requires the execution of a task c which is neither primitive nor decomposable into sub-tasks. Task c is defined as a constant of task x, and therefore the planner can achieve the goal by instantiating an execution of x and then inferring task c as complete. The solution found by the planner to this problem is given in Listing B.9.

Listing B.7: A PDDL description of an action that instantiates new executions of tasks.

```
(:action instantiate_execution
    :parameters (?t - task ?i - execution ?e - environment)
    :precondition(and
      (not
        (exists (?x)
          (has_task ?i ?x)))
      (not
        (exists (?x)
          (has_env ?i ?x))))
    :effect (and
      (has_task ?i ?t)
      (has_env ?i ?e)))
```

Listing B.8: A PROHOW check-list containing a constant, defined as a PDDL problem.

```
(define (problem sample)
  (:domain prohow)
  (:objects
   t b x c - task
   e ee - environment
   ex1 ex2 ex3 - execution)
  (:init
    (has_goal e t)
    (has_step t b )
    (has_method b x)
    (requires x c)
    (primitive x)
    (has_constant x c))
  (:goal (finished e) ))
```

Listing B.9: A plan generated by the Fast-Downward planner to solve the problem defined in Listing B.8 under the PDDL domain description of the PROHOW model with the primitive task extension.

```
(instantiate_execution x ex1 ee)
(infer_complete_from_constant c ee)
(infer_ready_by_requirements x ee)
(accomplish_task x ee ex2)
(accomplish_task x ee ex3)
(infer_ready_by_requirements t e)
(infer_ready_by_requirements b e)
(instantiate_sub_env ee e b x)
(infer_complete x ee)
(infer_complete_by_method b ee)
(infer_complete_by_sub_env b e)
(infer_complete_by_steps t e)
(infer_finished e)
```

Appendix C

SPARQL Queries

C.1 Queries to Compute Statistics

Listing C.1: SPARQL query to count the number of main sets of instructions

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT (COUNT (DISTINCT ?main) AS ?number)
WHERE{
  ?main rdf:type <http://w3id.org/prohow#instruction_set> .
}
```

Listing C.2: SPARQL query to count the number tasks which have steps

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT (COUNT (DISTINCT ?main) AS ?number)
WHERE{
  ?main prohow:has_step  ?s .
}
```

Listing C.3: SPARQL query to count the number of labelled requirements of the main instructions sets

```
PREFIX prohow: <http://w3id.org/prohow#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT (COUNT (DISTINCT ?r) AS ?number)
WHERE{
  ?main rdf:type <http://w3id.org/prohow#instruction_set> .
  ?main prohow:requires ?r .
  ?r rdfs:label ?l .
}
```

Listing C.4: SPARQL query to count the number of labelled steps

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT (COUNT (DISTINCT ?s) AS ?number)
WHERE{
  ?main prohow:has_step ?s .
  ?s rdfs:label ?l .
}
```

Listing C.5: SPARQL query to count the number of labelled methods

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT (COUNT (DISTINCT ?m) AS ?number)
WHERE{
  ?main prohow:has_method ?m .
  ?m rdfs:label ?l .
}
```

Listing C.6: SPARQL query to count the number of labelled entities

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT (COUNT (DISTINCT ?entity) AS ?number)
WHERE{
  ?entity rdfs:label ?l .
}
```

Listing C.7: SPARQL query to count the number of sets of instructions with multiple sets of requirements.

```
PREFIX prohow: <http://w3id.org/prohow#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT (COUNT(distinct ?main) AS ?number)
WHERE {
  ?main rdf:type prohow:instruction_set .
  ?main prohow:requires ?r .
  ?r prohow:has_step ?x .
 }
```

C.2 Queries Used in Algorithms

SPARQL queries can be defined to return sets of tasks. The SPARQL template defined in Listing C.8 can be used to define a number of queries about a task *t* by replacing the place-holder `%SUBJ%` by the URI of task *t*. In the resulting query, the place-holder `%REL%` can then be replaced by the URI of a PROHOW relation to obtain one of several different functions, each returning a set of tasks. The following functions are defined using symbol $\mapsto$ to denote the replacement of a place-holder by a URI.

- `get_steps`(t) *if* `%REL%` $\mapsto$ `prohow:has_step`
- `get_methods`(t) *if* `%REL%` $\mapsto$ `prohow:has_method`
- `get_requirements`(t) *if* `%REL%` $\mapsto$ `prohow:requires`
- `get_sufficient_requirements`(t) *if* `%REL%` $\mapsto$ `prohow:requires_one`

If place-holder `%SUBJ%` is replaced by the URI of an environment e, then Template C.8 can be used to define the following functions.

- `get_super_environments`(e) *if* `%REL%` $\mapsto$ `prohow:sub_environment_of`
- `get_goal`(e) *if* `%REL%` $\mapsto$ `prohow:has_goal`

Listing C.8: SPARQL template to find the set of objects related with relation r from subject s. Place-holder `%SUBJ%` must be replaced by the URI of s and place-holder `%REL%` with the URI of r.

```
SELECT ?o
WHERE {
 <%SUBJ%> <%REL%> ?o .
}
```

Query C.9 is the inverse of the previously mentioned one, and it can be used to discover the following functions if the URI of t is replaced by the place-holder `%OBJ%`:

- `get_effects`(t) *if* `%REL%` $\mapsto$ `prohow:has_method`
- `get_super_tasks`(t) *if* `%REL%` $\mapsto$ `prohow:has_step`

Listing C.9: SPARQL template to find the set of subjects s related with relation r to object o. Place-holder `%OBJ%` must be replaced by the URI of o and place-holder `%REL%` with the URI of r.

```
SELECT ?s
WHERE {
 ?s <%REL%> <%OBJ%> .
}
```

SPARQL query C.10 implements the function `get_task_ex_in_env`(t,e) by replacing the URI of e to place-holder %ENV% and the URI of t to place-holder %TASK%. This function returns one execution of task t in environment e if it exists.

Listing C.10: SPARQL template to find the set of tasks that have executions in environment e. Place-holder `%ENV%` must be replaced by the URI of e.

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT ?ex
WHERE {
  ?ex prohow:has_environment <%ENV%> .
  ?ex prohow:has_task <%TASK%> .
} LIMIT 1
```

SPARQL query C.11 implements the function `get_tasks_in_env`(e) by replacing the URI of e to place-holder %ENV%.

Listing C.11: SPARQL template to find the set of tasks that have executions in environment e. Place-holder `%ENV%` must be replaced by the URI of e.

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT ?task
WHERE {
  ?ex prohow:has_environment <%ENV%> .
  ?ex prohow:has_task ?task .
}
```

The function `get_completed_tasks_in_env`(e) is implemented by SPARQL query C.12 by replacing the URI of e to place-holder %ENV%.

Listing C.12: SPARQL template to find the set of tasks which are complete in environment e. Place-holder `%ENV%` must be replaced by the URI of e.

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT ?task
WHERE {
  ?ex prohow:has_environment <%ENV%> .
  ?ex prohow:has_task ?task .
  ?ex prohow:has_result prohow:complete .
}
```

The function `get_sub_environments_in_env`(e) is implemented by SPARQL query C.13 by replacing the URI of e to place-holder %ENV%.

Listing C.13: SPARQL template to find the set of pairs t, a such that task t is scheduled to be completed in sub-environments a of e. Place-holder `%ENV%` must be replaced by the URI of e.

```
PREFIX prohow: <http://w3id.org/prohow#>
SELECT ?task ?env
WHERE {
  ?env prohow:sub_environment_of <%ENV%> .
  ?env prohow:has_goal ?task .
}
```

C.3 Update Queries Used in Algorithms

Databases are often dynamic objects which can constantly change. This is especially true for databases recording the state of the execution of a process. Every time the state of the execution changes, the database must change as well. The most simple change that can affect an RDF dataset is the addition of a triple.

The SPARQL Update template C.14 inserts a triple $< s, p, o >$ in the dataset if the place-holders %S%, %P% and %O% are replaced, respectively, by the URIs of s, p and o. This insertion of a triple defines the function `assert_triple`(s, p, o).

Listing C.14: SPARQL Update query to insert a triple.

```
INSERT DATA {
  <%S%> <%P%> <%O%> .
}
```

The `update_database`() function, that can be used to materialise inferred facts into a PROHOW dataset, is defined by the sequential execution of the queries C.15, C.16, C.17 and C.18.

Listing C.15: SPARQL Update query to declare tasks complete according to Formula 13.

```
PREFIX prohow: <http://w3id.org/prohow#>
INSERT {
  ?ex prohow:has_result prohow:complete .
}
WHERE {
  ?ex prohow:has_environment ?env .
  ?ex prohow:has_task ?task .
  ?env prohow:has_goal ?task .
  ?task prohow:has_method ?method .
  ?m_ex prohow:has_environment ?env .
  ?m_ex prohow:has_task ?method .
  ?m_ex prohow:has_result prohow:complete .
}
```

Listing C.16: SPARQL Update query to declare tasks complete according to Formula 17.

```
PREFIX prohow: <http://w3id.org/prohow#>
INSERT {
  ?ex prohow:has_result prohow:complete .
}
WHERE {
  ?ex prohow:has_environment ?env .
  ?ex prohow:has_task ?task .
  ?sub_env prohow:sub_environment_of ?env .
  ?sub_env prohow:has_goal ?task .
  ?sub_ex prohow:has_environment ?sub_env .
  ?sub_ex prohow:has_task ?task .
  ?sub_ex prohow:has_result prohow:complete .
}
```

Listing C.17: SPARQL Update query to declare tasks complete according to Formula 12.

```
PREFIX prohow: <http://w3id.org/prohow#>
INSERT  {
  ?ex prohow:has_result prohow:complete .
}
WHERE {
  ?ex prohow:has_environment ?env .
  ?ex prohow:has_task ?task .
  ?task prohow:has_step ?step .
  MINUS {
    ?ex prohow:has_environment ?env .
    ?ex prohow:has_task ?task .
    ?task prohow:has_step ?s .
    FILTER NOT EXISTS {
      ?exs prohow:has_task ?s .
      ?exs prohow:has_environment ?env .
      ?exs prohow:has_result prohow:complete .
    }
  }
}
```

Listing C.18: SPARQL Update query to declare tasks complete according to Formula 19.

```
PREFIX prohow: <http://w3id.org/prohow#>
INSERT  {
  ?ex prohow:has_result prohow:complete .
}
WHERE {
  ?ex prohow:has_environment ?env .
  ?ex prohow:has_task ?task .
  ?task prohow:binds_to ?binding .
  ?ex_b prohow:has_task ?binding .
  ?ex_b prohow:result prohow:complete .
  ?ex_b prohow:has_environment ?env_b .
  ?env_b (prohow:sub_environment_of |
          ^prohow:sub_environment_of)* ?env .
}
```

Appendix D

Algorithms

This Chapter of the Appendix lists the algorithms that I implemented to computationally solve the competency questions `Q1` to `Q8` defined in Section 4.7.3. These algorithms, which will be listed in details later, are the following:

Q1 : `get_requirement_sets`(*task x*) returns a set of sets of tasks;

Q2 : `get_decomposition_sets`(*task x*) returns a set of sets of tasks;

Q3 : `get_tasks_in_env`(*environment e*) returns a set of tasks;

Q4 : `get_completed_tasks_in_env`(*environment e*) returns a set of tasks;

Q5 : `get_ready_tasks_in_env`(*environment e*) returns a set of tasks;

Q6 : `get_sub_environments_in_env`(*environment e*) returns a set of pairs of tasks and environments;

Q7 : `get_next_tasks_to_accomplish`(*environment e*) returns a set of tasks;

Q8 : `create_checklists`(*task x*) returns a set of lists of sets of tasks.

These algorithms make use of the following functions defined as SPARQL queries in Section C.2 of the Appendix.

- `get_steps`(t)
- `get_methods`(t)
- `get_requirements`(t)

- `get_sufficient_requirements`(t)
- `get_super_environments`(e)
- `get_goal`(e)
- `get_effects`(t)
- `get_super_tasks`(t)
- `get_task_ex_in_env`(t, e)
- `get_tasks_in_env`(e)
- `get_completed_tasks_in_env`(e)
- `get_sub_environments_in_env`(e)

In Section C.2 these functions are presented as SPARQL templates, namely SPARQL queries with optional place-holders that need to be instantiated before the query can be considered complete. Each query is identified as a unique function name that instantiates the query and returns its results.

D.1 Algorithms to Query a PROHOW Dataset

In order to answer competency question `Q1`, Algorithm 1 retrieves the set of sets of entities which are needed to declare a task x ready for execution.

Algorithm 1 An algorithm to solve competency question `Q1`.

```
1:  procedure get_requirement_sets($x$)
2:      $R_{and} \leftarrow$ get_requirements($x$)
3:      $R_{or} \leftarrow$ get_sufficient_requirements($x$)
4:      $R_{sets} \leftarrow$ empty set of sets
5:      if $R_{and}$ contains at least one element then
6:          $R_{sets}$ add element $R_{and}$
7:      for $\forall r . r \in R_{or}$ do
8:          $R_{new} \leftarrow$ empty set
9:          $R_{new}$ add element $r$
10:         $R_{sets}$ add element $R_{new}$
11:     return $R_{sets}$
```

Competency question Q2 can be answered by Algorithm 2, which retrieves the set of sets of entities which can sufficiently decompose a task x into a different but equivalent set of tasks.

Algorithm 2 An algorithm to solve competency question Q2.

```
 1: procedure get_decomposition_sets($x$)
 2:     $D_{and} \leftarrow$ get_steps($x$)
 3:     $D_{or} \leftarrow$ get_methods($x$)
 4:     $D_{sets} \leftarrow$ empty set of sets
 5:     if $D_{and}$ contains at least one element then
 6:         $D_{sets}$ add element $D_{and}$
 7:     for $\forall d \,.\, d \in D_{or}$ do
 8:         $D_{new} \leftarrow$ empty set
 9:         $D_{new}$ add element $d$
10:         $D_{sets}$ add element $D_{new}$
11:     return $D_{sets}$
```

Competency question Q3 and Q4 can be directly answered by a SPARQL query following the templates of Listings C.11 and C.12, respectively. Competency question Q5 can be answered by Algorithm 3 and the utility Algorithms 4 and 5. Algorithm 3 returns the set of tasks in environment e which have all of their requirements completed. Algorithm 4 returns the boolean value *True* if task t is ready to be executed in environment e, and *False* otherwise. Algorithm 5 returns the boolean value *True* if environment e is active, and *False* otherwise.

Algorithm 3 An algorithm to solve competency question Q5.

```
1: procedure get_ready_tasks_in_env($e$)
2:     $R \leftarrow$ empty set
3:     $T \leftarrow$ get_tasks_in_env($e$)
4:     for $\forall t \,.\, t \in T$ do
5:         if is_task_ready_in_env($t, e$) then
6:             $R$ add element $t$
7:     return $R$
```

Algorithm 4 An algorithm to determine whether a task t is ready to be executed in an environment e.

```
1: procedure is_task_ready_in_env($t, e$)
2:     if is_environment_active($e$) $==$ $False$ then
3:         return $False$
4:     $S \leftarrow$ get_super_tasks($t$)
5:     if $S$ is not empty then
6:         $B \leftarrow False$
7:         for $s \in S$ do
8:             if $B = False$ then
9:                 $B \leftarrow$ is_task_ready_in_env($s, e$)
10:        if $B = False$ then
11:            return $False$
12:    $R_{all} \leftarrow$ get_requirement_sets($x$)
13:    if $R_{all}$ is empty then
14:        return $True$
15:    $C \leftarrow$ get_completed_tasks_in_env($e$)
16:    for $\forall R . R \in R_{all}$ do
17:        if $R \in C$ then
18:            return $True$
19:    return $False$
```

Algorithm 5 An algorithm to determine whether an environment a is active.

```
1: procedure is_environment_active($e$)
2:     $E_{super} \leftarrow$ get_super_environments($e$)
3:     if $E_{super}$ is empty then
4:         return $True$
5:     $g \leftarrow$ get_goal($e$)
6:     for $a \in E_{super}$ do
7:         if is_task_ready_in_env($g, a$) $= True$ then
8:             return $True$
9:     return $False$
```

Competency question Q6 can be directly answered by a SPARQL query using the template of Listings C.13. Competency question Q7 can be answered by Algorithm 6, to discover the set of tasks within an environment *e* which can be accomplished next to progress toward the completion of *e*. Algorithm 7, using Algorithm 8, answers the recursive version of this question by considering also the sub-environments of *e*. This last algorithm returns a map containing a set of related environments as keys. Each one of these environments maps to the set of tasks that can be accomplished in that environment to progress its execution.

Algorithm 6 An algorithm to solve competency question `Q7` without considering sub-environments.

```
1: procedure get_next_tasks_to_accomplish($e$)
2:     $R \leftarrow$ get_ready_tasks_in_env($e$)
3:     $C \leftarrow$ get_completed_tasks_in_env($e$)
4:     $R_{\neg C} \leftarrow R \setminus C$ (elements in $R$ not in $C$)
5:     return $R_{\neg C}$
```

Algorithm 7 An algorithm to solve competency question `Q7` considering sub-environments.

```
1: procedure get_next_tasks_to_accomplish_recursive($e$)
2:     $A \leftarrow$ empty map
3:     expand_map_with_tasks_to_accomplish_in_subenv($e,A$)
4:     return $A$
```

Algorithm 8 Recursive part of algorithm 7.

```
1: procedure expand_map_with_tasks_to_accomplish_in_subenv($e,A$)
2:     $A[e] \leftarrow$ get_next_tasks_to_accomplish($e$)
3:     $E \leftarrow$ get_sub_environments_in_env($e$)
4:     for $\forall s . s \in E$ do
5:         $t \leftarrow$ (first) task element of pair $s$
6:         if $t \in A[e]$ then
7:             $a \leftarrow$ (second) environment element of pair $s$
8:             expand_map_with_tasks_to_accomplish_in_subenv($a,A$)
```

D.2 Algorithms to Generate PROHOW Check-Lists

To answer question `Q8`, I will show how all the non-redundant check-lists to achieve a given goal can be procedurally generated. In this context, a non-redundant check-list is a check-list such that the removal of any of its tasks either prevents the goal to be achieved, or it forces a super-task to be completed instead of one of its sub-tasks. While the possibility of generating all the non-redundant check-lists can be useful for simple task networks, large task networks comprising many different levels of decomposition and choice points might make this generation effort computationally inefficient. In this last case, particular rules and heuristics can be implemented to discard unwanted check-list configurations.

In these algorithms, an unordered check-list is uniquely identified by the set of tasks it contains. Two check-lists containing the same set of tasks can be said to be equivalent. For example, the set $\{a, b\}$ might represent the a check-list containing tasks a and b to complete task t. This represents the logical formula $t \leftarrow a \wedge b$. Multiple options are represented by a set of sets, or Alternative Sets (AS). For example, a task t might be decomposable by any sets of tasks in the AS $\{\{a, b\}, \{a, c, d\}\}$. This represents the logical formula $t \leftarrow (a \wedge b) \vee (a \wedge c \wedge d)$. AS represent options in disjunctive normal form (DNF). For computational purposes, it is convenient to be able to normalise the disjunction or the conjunction of two AS into a single AS. This is equivalent to performing a normalisation of their logical interpretation into DNF. The disjunction of two AS A and B normalised in DNF is simply the union of the two sets: $A \cup B$. The conjunction of two AS A and B can be normalised in DNF by Algorithm 9 using the distributive rule of replacement $X \wedge (Y \vee Z) \iff (X \wedge Y) \vee (X \wedge Z)$. This algorithm also exploits the fact that computational sets do not contain duplicate elements and that sets containing the same set of elements are considered equivalent.

Algorithm 9 An algorithm to merge the conjunction of two AS sets A and B into a single AS.

```
1: procedure merge_pair_of_AS($A, B$)
2:     if $A$ is empty then
3:         return $B$
4:     if $B$ is empty then
5:         return $A$
6:     $M \leftarrow$ empty set
7:     for $\forall A_{alternative} . A_{alternative} \in A$ do
8:         for $\forall B_{alternative} . B_{alternative} \in B$ do
9:             $M$ add element $A_{alternative} \cup B_{alternative}$
10:    return $M$
```

Algorithm 10 answers questions `Q8` by computing all the possible sets of tasks that would enable a task t to be ready for execution, and all the possible step decomposition of the tasks using Formula 12 and 13. The resulting sets of tasks are returned in a valid order (or partial) order unless an exception is raised. A valid order of a set of task is an order that does not violate the dependencies of its task. An exception can be raised by Algorithm 12 if a dependency deadlock has been discovered and no valid check-list can be generated. If no exception is raised, any list in the set returned by Algorithm 10 can be considered a valid answer to question `Q8`.

For example, a task t: "prepare milk coffee", might require either the ingredient a: "instant milk coffee powder" or both the ingredients b: "instant coffee powder" and c: "milk". In this example, Algorithm 10 would return the set of lists of sets $\{[\{a\},\{t\}],[\{b,c\},\{t\}]\}$, which can be translated into either a check-list where task a precedes task t, or a check-list where both tasks b and c, in any order, precede task t.

Decompositions of tasks using Formula 17 are not included in Algorithm 10 as they do not change a check-list by itself. More specifically, Formula 17 allows a task in a given check-list to be decomposed by one of its methods in a different but related check-list. In a check-list e, the decomposition of a task t with method m can be represented by defining another check-list m, sub-environment of e, generated by algorithm `create_checklists`(t) and containing method m.

Algorithm 10 An algorithm to get a set of check-lists to answer question `Q8` without considering task decomposition.

```
1: procedure create_checklists($t$)
2:     $U \leftarrow$ expand_checklist_requirements($\{t\}, \{\}$)
3:     $U_{methods} \leftarrow$ empty set of sets
4:     $M \leftarrow$ get_methods($t$)
5:     for $m \in M$ do
6:         for $S \in U$ do
7:             $S_m \leftarrow$ empty set
8:             $S_m$ add all entities in $S$
9:             $S_m$ add entity m
10:            $S_{m\_all} \leftarrow$ expand_checklist_requirements($S_m, S$)
11:            $S_{m\_all}$ add all in expand_checklist_steps($S_{m\_all}, \{t\}$)
12:            $U_{methods}$ add all sets in $S_{m\_all}$
13:    $U \leftarrow U \cup$ expand_checklist_steps($U, \{\}$) $\cup U_{methods}$
14:    $O \leftarrow$ empty set of lists of sets
15:    for $S \in U$ do
16:        $O$ add element order_task_set($S$)
17:    return $O$
```

Algorithm 11 An algorithm to expand the requirements of the set of tasks S of a checklist excluding the tasks in A.

```
1: procedure expand_checklist_requirements($(S, A)$)
2:     $A_{new} \leftarrow$ empty set
3:     $A_{new}$ add all elements in $A$
4:     $S_{new} \leftarrow$ empty set
5:     $S_{new}$ add all elements in $S$
6:     $T \leftarrow$ empty set of sets
7:     $T \leftarrow$ add element $S_{new}$
8:     for $\forall t \,.\, t \in S_{new}$ do
9:         if $t \notin A_{new}$ then
10:            $R \leftarrow$ get_requirement_sets($(t)$)
11:            if $R$ is not empty then
12:                $T \leftarrow$ merge_pair_of_AS($(T, R)$)
13:    $A_{new}$ add all elements in $S$
14:    for $\forall P \,.\, P \in T$ do
15:        if $\neg P \subset A_{new}$ then
16:            $P_{expanded} \leftarrow$ expand_checklist_requirements($(P, A_{new})$)
17:            $T \leftarrow T$ remove element $P$
18:            $T \leftarrow T \cup P_{expanded}$
19:    return $T$
```

Algorithm 12 An algorithm to order a set of tasks S.

```
1: procedure order_task_set($S$)
2:     $L \leftarrow$ empty list
3:     $S_{left} \leftarrow$ empty set
4:     $S_{left}$ add all entities in $S$
5:     while $S_{left}$ is not empty do
6:         $A \leftarrow$ empty set
7:         $A$ add all entities in $L$
8:         $O \leftarrow$ empty set
9:         $O_{is\_empty} \leftarrow True$
10:        for $t \in S_{left}$ do
11:            $R_{satisfied} \leftarrow False$
12:            $R \leftarrow$ get_requirement_sets($t$)
13:            if $R$ is empty then
14:                $R_{satisfied} \leftarrow True$
15:            else
16:                for $r \in R$ do
17:                    if $r \subset A$ then
18:                        $R_{satisfied} \leftarrow True$
19:            if $R_{satisfied} = True$ then
20:                $O$ add element $t$
21:                $O_{is\_empty} \leftarrow False$
22:        if $O_{is\_empty} = True$ then
23:            throw deadlock exception
24:        $L$ append element $O$
25:        $S_{left} \leftarrow S_{left} \backslash O$
26:    return $L$
```

Algorithm 13 expands a SA with all possible decompositions of its tasks into sub-steps. This algorithm iterates over a set of tasks trying to decompose them into steps. When new steps T_{steps} of a task are found they are included in a new set of task $S_{pre} \cup T_{steps}$. This new set is then expanded with all the requirements of the new tasks using Algorithm 11.

Algorithm 13 An algorithm to expand a check-list with all the possible step decompositions, except for the decompositions of the tasks in K.

```
1: procedure expand_checklist_steps($S_{previous\_all}, K$)
2:     $S_{new\_all} \leftarrow$ empty set of sets
3:     for $S_{pre} \in S_{previous\_all}$ do
4:         for $t \in S_{pre}$ do
5:             if $t \notin K$ then
6:                 $T_{steps} \leftarrow$ get_steps($t$)
7:                 if $T_{steps}$ is not empty $\wedge \neg T_{steps} \subset S_{pre}$ then
8:                     $S_{new\_exp} \leftarrow$ expand_checklist_requirements($S_{pre} \cup T_{steps}, S_{pre}$)
9:                     $S_{new\_all}$ add all elements in $S_{new\_exp}$
10:                    $S_{new\_all}$ add all elements in expand_checklist_steps($S_{new\_exp}, K \cup \{t\}$)
11:    return $S_{new\_all}$
```

D.3 Algorithms to Materialise and Validate Algorithms

These algorithms make use of the `assert_triple`(s,p,o) and `update_database`() functions defined in Section C.3 of the Appendix as SPARQL Update queries.

During the execution of a process, it might be possible to infer whether a certain task has been completed by completing all of its steps or one of its methods. For example, update queries C.15, C.16, C.17 and C.18 assert executions as complete if the completion of their task can be inferred from, respectively, Formulas 13, 17, 12 and 19. When such inference is made, it is useful to record it in the knowledge base to avoid the overhead of re-computing such inferences every time a query is made. The algorithms given in this section assume the materialisation of such inferred facts as explicit statements in the knowledge base.

Whenever the RDF database is modified, an update operation `update_database()`, consisting of the queries C.15, C.16, C.17 and C.18 needs to be performed before any other query is computed. If these queries result in a modification of the database, or more specifically when new triples are added, a subsequent update needs to be performed. Multiple iterations of the update procedure might be necessary because each of those queries only apply the logical inferences to the statements in the dataset, and not to the inferred statements as well. In order to determine whether the update procedure has modified the database multiple approaches are possible. For example, it is possible to count the number of triples in the database before and after the modifications.

Each set of tasks S generated by calling the function `create_checklists`(t) that I previously defined can be represented as a PROHOW check-list (or environment) e in RDF using function `materialise_checklist`(S,t) defined in Algorithm 14. This function returns the URI of e upon completion, and it makes use of the function `mint_URI`$()$, a function that mints a new URI. This function will not be defined formally as it can be implemented in many different ways which depend on the context where it is to be used. One possible approach to minting URIs was described in Section 2.1.1.

Algorithm 14 An algorithm to materialise a new check-list, defined as a set of tasks, or a list of sets of tasks S with goal t.

```
1:  procedure materialise_checklist($S,t$)
2:      $e \leftarrow$ mint_URI()
3:      assert_triple($e$,“prohow:has_goal”,$t$)
4:      if $S$ is a list of sets then
5:          $S \leftarrow$ union of all sets in $S$
6:      for $x \in S$ do
7:          $ex \leftarrow$ mint_URI()
8:          assert_triple($ex$,“prohow:has_environment”,$e$)
9:          assert_triple($ex$,“’prohow:has_task”,$x$)
10:     return $e$
```

The execution of a check-list e with goal t generated by Algorithm 10 and instantiated by Algorithm 14 can be simulated by Algorithm 15 to verify whether it is feasible, namely if it does not violate dependency constraints, and whether it does actually accomplish goal t. This algorithm makes use of the function `update_database`$()$, previously defined as the subsequent call of the SPARQL Update queries C.15, C.16, C.17 and C.18. This update function ensures that tasks are declared complete in the environment as soon as they can be inferred to be so.

Algorithm 15 An algorithm to validate a check-list e having a list of tasks L and goal t.

```
1:  procedure validate_checklist($L, e, t$)
2:      for $S \in L$ do
3:          for $x \in S$ do
4:              if is_task_ready_in_env($x, e$) $= False$ then
5:                  return $False$
6:              $ex \leftarrow$ get_task_ex_in_env($x, e$)
7:              if $ex$ is not null then
8:                  assert_triple($ex$,“prohow:has_result”,“prohow:complete”)
9:                  update_database()
10:     $C \leftarrow$ get_completed_tasks_in_environment($e$)
11:     if $t \in C$ then
12:         return $True$
13:     else
14:         return $False$
```

Algorithm 15 can be easily extended into Algorithm 16 to incorporate the concept of a *primitive tasks set* (PTS). Inspired by the notion of primitive tasks in the AP field, a PTS represents the basic capabilities of an agent (or multiple agents), or more specifically the set of tasks that the agent can directly accomplish without decomposing tasks into sub-tasks. Given a PTS A, Algorithm 16 returns true only if the analysed check-list can be completed by only accomplishing tasks in A. Although more computationally efficient algorithms are possible, the validation of Algorithm 16 can be immediately applied to all the possible check-lists computed by Algorithm 10 to select only those, if any, that an agent could complete.

Algorithm 16 An algorithm to validate whether a primitive task set A is sufficient to complete a check-list e having a list of tasks L and goal t.

```
1: procedure validate_checklist_with_primitives($A, L, e, t$)
2:     for $S \in L$ do
3:         for $x \in S \cap A$ do
4:             if is_task_ready_in_env($x, e$) = True then
5:                 $ex \leftarrow$ get_task_ex_in_env($x, e$)
6:                 if $ex$ is not null then
7:                     assert_triple($ex$,“prohow:has_result”,“prohow:complete”)
8:                     update_database()
9:     $C \leftarrow$ get_completed_tasks_in_environment($e$)
10:    if $t \in C$ then
11:        return True
12:    else
13:        return False
```

Appendix E

Linguistic Resources

The chapter provides additional details on the resources used by the interlinking system presented in Chapter 6.

E.1 Units of Measure

These units of measure have been generated by combining standard lists of units of measure with additional units frequently found in the dataset. For example, the word "copy" is not a standard unit of measure, although it is commonly used to define a relative measure.

- tablespoon
- group
- splash
- teaspoon
- amount
- cup
- punnet
- jar
- bucket
- m
- bar
- bag
- variety
- gallon
- copy
- yard
- pair
- ft
- grade
- bunch
- sheet
- box
- feet
- foot
- source
- slice
- pinch
- lot
- copies
- number
- can
- glass
- ml
- bottle
- kind
- piece
- tsp
- g
- capful
- tbsp

- ball
- packet
- inch
- inches
- oz
- quart
- lbs
- ounce
- drop
- liter
- l

E.2 Creation Verbs

- make
- build
- create
- assemble
- cook
- prepare
- do
- bake
- brew
- install
- find
- calculate
- compute
- setup
- construct
- generate
- produce
- become
- get
- organize
- perform
- catch
- capture
- buy
- acquire
- purchase
- obtain
- grow
- harvest

E.3 Type Preserving Adjectives

- ground
- melted
- fresh
- sliced
- grated
- diced

Bibliography

Addis, A. and Borrajo, D. (2011). From Unstructured Web Knowledge to Plan Descriptions. In *Information Retrieval and Mining in Distributed Environments*, volume 324 of *Studies in Computational Intelligence*, pages 41–59. Springer Berlin Heidelberg.

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web Services*, pages 123–149. Springer Berlin Heidelberg, Berlin, Heidelberg.

Álvarez, M., Pan, A., Raposo, J., Bellas, F., and Cacheda, F. (2008). Extracting lists of data records from semi-structured web pages. *Data & Knowledge Engineering*, 64(2):491 – 509.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer Berlin Heidelberg.

Azaria, A., Krishnamurthy, J., and Mitchell, T. M. (2016). Instructable Intelligent Personal Agent. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2681–2689. AAAI Press.

Baader, F., Horrocks, I., and Sattler, U. (2008). Chapter 3 Description Logics. In Frank van Harmelen, V. L. and Porter, B., editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 135 – 179. Elsevier.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL Web Ontology Language Reference. *W3C Recommendation February*, 10.

Beetz, M., Langer, H., and Nyga, D. (2015). *Planning Everyday Manipulation Tasks –*

Prediction-based Transformation of Structured Activity Descriptions, pages 63–83. Springer Fachmedien Wiesbaden, Wiesbaden.

Bellinger, G., Castro, D., and Mills, A. (2004). Data, Information, Knowledge, and Wisdom. Available from http://www.systems-thinking.org/dikw/dikw.htm (retrieved on 17/10/2017).

Bizer, C., Heath, T., and Berners-Lee, T. (2009a). Linked Data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22.

Bizer, C., Volz, J., Kobilarov, G., and Gaedke, M. (2009b). Silk - A Link Discovery Framework for the Web of Data. In *18th International World Wide Web Conference*.

Branavan, S. R. K., Zettlemoyer, L. S., and Barzilay, R. (2010). Reading Between the Lines: Learning to Map High-level Instructions to Commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1268–1277, Stroudsburg, PA, USA. Association for Computational Linguistics.

Brickley, D. and Guha, R. (2014). RDF Schema 1.1. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

Carothers, G. and Prud'hommeaux, E. (2014). RDF 1.1 Turtle. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-turtle-20140225/.

Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999). What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, 14(1):20–26.

Chernov, A., Lagos, N., Gallé, M., and Sándor, A. (2016). Enriching How-to Guides by Linking Actionable Phrases. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 939–944, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27(3):289 – 350.

Constantinescu, I., Faltings, B., and Binder, W. (2004). Large scale, Type-Compatible Service Composition. In *Proceedings. IEEE International Conference on Web Services, 2004.*, pages 506–513.

Corkill, D. D. (1991). Blackboard Systems. *AI Expert*, 6(9):40–47.

Department, M. G. and Gruninger, M. (1996). Designing and Evaluating Generic Ontologies. In *In Proceedings of the 12th European Conference of Artificial Intelligence*.

Eickhoff, C. and de Vries, A. P. (2013). Increasing Cheat Robustness of Crowdsourcing Tasks. *Information Retrieval*, 16(2):121–137.

Erol, K., Hendler, J., and Nau, D. S. (1994a). HTN Planning: Complexity and Expressivity. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, AAAI'94, pages 1123–1128, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Erol, K., Hendler, J., and Nau, D. S. (1994b). UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, AIPS'94, pages 249–254. AAAI Press.

Faria, C., Serra, I., and Girardi, R. (2014). A Domain-Independent Process for Automatic Ontology Population from Text. *Science of Computer Programming*, 95:26 – 43. Special Issue on Systems Development by Means of Semantic Technologies.

Fellegi, I. P. and Sunter, A. B. (1969). A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210.

Fensel, D., Facca, F., Simperl, E., and Toma, I. (2011). Triple Space Computing for Semantic Web Services. In *Semantic Web Services*, pages 219–249. Springer Berlin Heidelberg.

Fernández-López, M., Gómez-Pérez, A., and Juristo, N. (1997). METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*. American Asociation for Artificial Intelligence. Ontology Engineering Group ? OEG.

Fukazawa, Y. and Ota, J. (2010). Automatic Modeling of User's Real World Activities from the Web for Semantic IR. In *Proceedings of the 3rd International Semantic Search Workshop*, pages 5:1–9.

Gangemi, A. (2013). *A Comparison of Knowledge Extraction Tools for the Semantic Web*, pages 351–366. Springer Berlin Heidelberg, Berlin, Heidelberg.

Geffner, H. and Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning: Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers, 1st edition.

Giaretta, P. and Guarino, N. (1995). Ontologies and Knowledge Bases Towards a Terminological Clarification. *Towards very large knowledge bases: knowledge building & knowledge sharing*, 25:32.

Grüninger, M. and Fox, M. S. (1995). *The Role of Competency Questions in Enterprise Engineering*, pages 22–31. Springer US, Boston, MA.

Grüninger, M. and Menzel, C. (2003). The Process Specification Language (PSL) Theory and Applications. *AI Magazine*, 24(3):63–74.

Guarino, N. and Welty, C. A. (2009). *An Overview of OntoClean*, pages 201–220. Springer Berlin Heidelberg, Berlin, Heidelberg.

Guha, R. V., Brickley, D., and Macbeth, S. (2016). Schema.Org: Evolution of Structured Data on the Web. *Commun. ACM*, 59(2):44–51.

Guy, I., Ronen, I., and Raviv, A. (2011). Personalized Activity Streams: Sifting Through the "River of News". In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 181–188, New York, NY, USA. ACM.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorer Newsletter*, 11(1):10–18.

Hassan, M. M., Speck, R., and Ngonga Ngomo, A.-C. (2015). *Using Caching for Local Link Discovery on Large Data Sets*, pages 344–354. Springer International Publishing, Cham.

Hayes, P. (2004). RDF Semantics. W3C Recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

Heath, T., Bizer, C., and Hendler, J. (2011). *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, 1st edition.

Herman, I., Sporny, M., Adida, B., and Birbeck, M. (2015). RDFa 1.1 Primer - Third Edition. W3C Note, W3C. http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.

Hickson, I., Lie, H. W., Bos, B., and Çelik, T. (2011). Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C Recommendation, W3C. http://www.w3.org/TR/2011/REC-CSS2-20110607.

Japkowicz, N. and Stephen, S. (2002). The Class Imbalance Problem: A Systematic Study. *Intell. Data Anal.*, 6(5):429–449.

Jones, D., Bench-Capon, T., and Visser, P. (1998). Methodologies for Ontology Development. In *IFIP World Computer Congress*, pages 62–75.

Jung, Y., Ryu, J., min Kim, K., and Myaeng, S.-H. (2010). Automatic Construction of a Large-Scale Situation Ontology by Mining How-To Instructions from the Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(2-3):110–124.

Ke, S.-R., Thuc, H. L. U., Lee, Y.-J., Hwang, J.-N., Yoo, J.-H., and Choi, K.-H. (2013). A Review on Video-Based Human Activity Recognition. *Computers*, 2(2):88–131.

Keil, D. and Goldin, D. (2006). *Indirect Interaction in Environments for Multi-agent Systems*, pages 68–87.

Kıcıman, E. and Richardson, M. (2015). Towards Decision Support and Goal Achievement: Identifying Action-Outcome Relationships From Social Media. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 547–556, New York, NY, USA. ACM.

Kiddon, C., Ponnuraj, G. T., Zettlemoyer, L. S., and Choi, Y. (2015). *Mise en Place: Unsupervised Interpretation of Instructional Recipes*, pages 982–992. Association for Computational Linguistics (ACL).

Kim, E., Helal, S., and Cook, D. (2010). Human Activity Recognition and Pattern Discovery. *Pervasive Computing, IEEE*, 9(1):48–53.

Krötzsch, M., Patel-Schneider, P., Parsia, B., Rudolph, S., and Hitzler, P. (2012). OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

Langegger, A. (2008). Virtual Data Integration on the Web: Novel Methods for Accessing Heterogeneous and Distributed Data with Rich Semantics. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 559–562, New York, NY, USA. ACM.

Lee, J., Gruninger, M., Jin, Y., Malone, T., Tate, A., and Yost, G. (1998). The Process Interchange Format and Framework. *The Knowledge Engineering Review*, 13:91–120.

Lenzerini, M. (2002). Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 233–246, New York, NY, USA. ACM.

Liu, X. Y., Wu, J., and Zhou, Z. H. (2009). Exploratory Undersampling for Class-Imbalance Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550.

Luz, N., Pereira, C., Silva, N., Novais, P., Teixeira, A., and Oliveira e Silva, M. (2014). An Ontology for Human-Machine Computation Workflow Specification. In *Hybrid Artificial Intelligence Systems*, volume 8480 of *Lecture Notes in Computer Science*, pages 49–60. Springer International Publishing.

Malmaud, J., Wagner, E. J., Chang, N., and Murphy, K. (2014). Cooking with Semantics. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 33–38.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D. L., Sirin, E., and Srinivasan, N. (2007). Bringing Semantics to Web Services with OWL-S. *World Wide Web*, 10(3):243–277.

Maynard, D., Li, Y., and Peters, W. (2008). NLP Techniques for Term Extraction and Ontology Population. In *Proceedings of the 2008 Conference on Ontology Learning and Population: Bridging the Gap Between Text and Knowledge*, pages 107–127, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical Report TR-98-003, Yale Center for Computational Vision and Control.

Michalowski, M., Ambite, J. L., Thakkar, S., Tuchinda, R., Knoblock, C. A., and Minton, S. (2004). Retrieving and Semantically Integrating Heterogeneous Data from the Web. *IEEE Intelligent Systems*, 19(3):72–79.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Minder, P. and Bernstein, A. (2012). CrowdLang: A Programming Language for the Systematic Exploration of Human Computation Systems. In Aberer, K., Flache, A., Jager, W., Liu, L., Tang, J., and Guret, C., editors, *Social Informatics*, volume 7710 of *Lecture Notes in Computer Science*, pages 124–137. Springer Berlin Heidelberg.

Minker, J. (1982). *On Indefinite Databases and the Closed World Assumption*, pages 292–308. Springer Berlin Heidelberg, Berlin, Heidelberg.

Montiel-Ponsoda, E., Vila-Suero, D., Villazón-Terrazas, B., Dunsire, G., Rodríguez, E. E., and Gómez-Pérez, A. (2011). Style Guidelines for Naming and Labeling Ontologies in the Multilingual Web. In *Proceedings of the 2011 International Conference on Dublin Core and Metadata Applications*, DCMI'11, pages 105–115. Dublin Core Metadata Initiative.

Murray-Rust, D. and Robertson, D. (2014). LSCitter: Building Social Machines by Augmenting Existing Social Networks with Interaction Models. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 875–880, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Myaeng, S.-H., Jeong, Y., and Jung, Y. (2013). Experiential Knowledge Mining. *Foundations and Trends in Web Science*, 4(1):71–82.

Nentwig, M., Hartung, M., Ngomo, A. N., and Rahm, E. (2017). A survey of current Link Discovery frameworks. *Semantic Web*, 8(3):419–436.

Ngomo, A.-C. N. and Auer, S. (2011). LIMES: A Time-efficient Approach for Large-scale Link Discovery on the Web of Data. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2312–2317. AAAI Press.

Ngonga Ngomo, A.-C. (2013). *ORCHID – Reduction-Ratio-Optimal Computation of Geo-spatial Distances for Link Discovery*, pages 395–410. Springer Berlin Heidelberg, Berlin, Heidelberg.

Nguyen, K. and Ichise, R. (2013). SLINT+ Results for OAEI 2013 Instance Matching. In *Proceedings of the 8th International Conference on Ontology Matching - Volume 1111*, OM'13, pages 177–183, Aachen, Germany, Germany. CEUR-WS.org.

Nikolov, A., Uren, V., and Motta, E. (2007). KnoFuss: A Comprehensive Architecture for Knowledge Fusion. In *Proceedings of the 4th International Conference on Knowledge Capture*, K-CAP '07, pages 185–186, New York, NY, USA. ACM.

Noy, N. F. and Mcguinness, D. L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory.

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004). Coordination Artifacts: Environment-Based Coordination for Intelligent Agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pages 286–293. IEEE Computer Society.

Otero-Cerdeira, L., Rodrguez-Martnez, F. J., and Gmez-Rodrguez, A. (2015). Ontology Matching: A Literature Review. *Expert Systems with Applications*, 42(2):949 – 971.

Pareti, P. (2016). Distributed Linked Data as a Framework for Human-Machine Collaboration. In Hartig, O., Sequeda, J., and Hogan, A., editors, *Proceedings of the 7th International Workshop on Consuming Linked Data (COLD)*, number 1666 in CEUR Workshop Proceedings, Aachen.

Pareti, P., Klein, E., and Barker, A. (2014a). A Semantic Web of Know-how: Linked Data for Community-centric Tasks. In *Proceedings of the 23rd International Conference on World Wide Web Companion*, pages 1011–1016.

Pareti, P., Klein, E., and Barker, A. (2015). A Linked Data Scalability Challenge: Concept Reuse Leads to Semantic Decay. In *Proceedings of the ACM Web Science Conference*, WebSci '15, pages 7:1–7:5, New York, NY, USA. ACM.

Pareti, P., Klein, E., and Barker, A. (2016). Linking Data, Services and Human Know-How. In *The Semantic Web. Latest Advances and New Domains*, ESWC 2016, pages 505–520. Springer International Publishing.

Pareti, P., Testu, B., Ichise, R., Klein, E., and Barker, A. (2014b). Integrating Know-How into the Linked Data Cloud. In *Knowledge Engineering and Knowledge Management*, volume 8876 of *Lecture Notes in Computer Science*, pages 385–396. Springer International Publishing.

Passant, A., Polleres, A., and Gearon, P. (2013). SPARQL 1.1 Update. W3C Recommendation, W3C. http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

Perkowitz, M., Philipose, M., Fishkin, K., and Patterson, D. J. (2004). Mining Models of Human Activities from the Web. In *Proceedings of the 13th international conference on World Wide Web*, pages 573–582.

Pexman, P. M., Hargreaves, I. S., Siakaluk, P. D., Bodner, G. E., and Pope, J. (2008). There are many ways to be rich: Effects of three measures of semantic richness on visual word recognition. *Psychonomic Bulletin & Review*, 15(1):161–167.

Poveda-Villalón, M., Gómez-Pérez, A., and Suárez-Figueroa, M. C. (2014). OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *Int. J. Semant. Web Inf. Syst.*, 10(2):7–34.

Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC., Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63.

Quilitz, B. and Leser, U. (2008). Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, volume 5021 of *LNCS*, pages 524–538.

Quinn, A. J. and Bederson, B. B. (2011). Human Computation: A Survey and Taxonomy of a Growing Field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA. ACM.

Quirk, C., Mooney, R., and Galley, M. (2015). 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 and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 878–888. Association for Computational Linguistics.

Raimond, Y. and Schreiber, G. (2014). RDF 1.1 Primer. W3C Note, W3C. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

Ren, Y., Parvizi, A., Mellish, C., Pan, J. Z., van Deemter, K., and Stevens, R. (2014). *Towards Competency Question-Driven Ontology Authoring*, pages 752–767. Springer International Publishing, Cham.

Ricci, A., Omicini, A., Viroli, M., Gardelli, L., and Oliva, E. (2007). Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts. In *Environments for Multi-Agent Systems III*, volume 4389 of *LNCS*, pages 124–140.

Robertson, D. (2005). *A Lightweight Coordination Calculus for Agent Systems*, pages 183–197. Springer Berlin Heidelberg, Berlin, Heidelberg.

Robertson, S. (2004). Understanding Inverse Documant Frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520. Copyright - Copyright MCB UP Limited (MCB) 2004; Document feature - equations; references; Last updated - 2014-05-25; CODEN - JDOCAS.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106.

Ryman, A., Moreau, J.-J., Chinnici, R., and Weerawarana, S. (2007). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation, W3C. http://www.w3.org/TR/2007/REC-wsdl20-20070626.

Sangers, J., Frasincar, F., Hogenboom, F., and Chepegin, V. (2013). Semantic Web Service Discovery Using Natural Language Processing Techniques. *Expert Systems with Applications*, 40(11):4660 – 4671.

Schall, D. (2012). *Service-Oriented Crowdsourcing: Architecture, Protocols and Algorithms*, chapter Human-Provided Services, pages 31–58.

Schumacher, P., Minor, M., Walter, K., and Bergmann, R. (2012). Extraction of Procedural Knowledge from the Web: a comparison of two workflow extraction approaches. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 739–747, New York, NY, USA. ACM.

Seaborne, A. and Harris, S. (2013). SPARQL 1.1 Query Language. W3C Recommendation, W3C. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

Shadbolt, N. R., Smith, D. A., Simperl, E., Van Kleek, M., Yang, Y., and Hall, W. (2013). Towards a Classification Framework for Social Machines. In *Proc. of the 22nd Int. Conf. on World Wide Web*, WWW '13 Companion, pages 905–912.

Snell, J. and Prodromou, E. (2017). Activity Streams 2.0. W3C Recommendation, W3C. https://www.w3.org/TR/2017/REC-activitystreams-core-20170523/.

Song, S., Oh, H., Myaeng, S., Choi, S., Chun, H., Choi, Y., and Jeong, C. (2011). Procedural Knowledge Extraction on MEDLINE Abstracts. In *Active Media Technology*, volume 6890 of *Lecture Notes in Computer Science*, pages 345–354. Springer Berlin Heidelberg.

Stuckenschmidt, H., Vdovjak, R., Broekstra, J., and Houben, G. (2005). Towards Distributed Processing of RDF Path Queries. *International Journal of Web Engineering and Technology*, 2(2/3):207–230.

Suárez-Figueroa, M. C., Gómez-Pérez, A., and Fernández-López, M. (2012). *The NeOn Methodology for Ontology Engineering*, pages 9–34. Springer Berlin Heidelberg, Berlin, Heidelberg.

Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., and Wenke, D. (2002). *OntoEdit: Collaborative Ontology Development for the Semantic Web*, pages 221–235. Springer Berlin Heidelberg, Berlin, Heidelberg.

Sure, Y., Staab, S., and Studer, R. (2009). *Ontology Engineering Methodology*, pages 135–152. Springer Berlin Heidelberg, Berlin, Heidelberg.

Tenorth, M., Klank, U., Pangercic, D., and Beetz, M. (2011). Web-Enabled Robots. *Robotics Automation Magazine, IEEE*, 18(2):58–68.

Tenorth, M., Nyga, D., and Beetz, M. (2010). Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1486–1491.

Thimm, M., Gottron, T., Gröner, G., and Scherp, A. (2012). Linked Open Data: Are we Drowning in Information and Starving for Know-How? In *Proceedings of the Workshop "What will the Semantic Web look like 10 years from now?" at the 11th International Semantic Web Conference (ISWC'12).*

Umbrich, J., Hose, K., Karnstedt, M., Harth, A., and Polleres, A. (2011). Comparing data summaries for processing live queries over Linked Data. *World Wide Web*, 14(5):495–544.

Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 11:93–136.

Van Der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003). Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51.

Vrandečić, D. (2009). Ontology evaluation. In Staab, S. and Studer, R., editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 293–313. Springer Berlin Heidelberg.

Weibel, S. L., Jul, E., and Shafer, K. E. (1996). *PURLs: Persistent Uniform Resource Locators*. OCLC Online Computer Library Center.

Welty, C. and Murdock, J. W. (2006). *Towards Knowledge Acquisition from Information Extraction*, pages 709–722. Springer Berlin Heidelberg, Berlin, Heidelberg.

Winkler, W. E. (2006). Overview of Record Linkage and Current Research Directions. Technical report, BUREAU OF THE CENSUS.

Young, B., Sanderson, R., and Ciccarese, P. (2017). Web Annotation Vocabulary. W3C Recommendation, W3C. https://www.w3.org/TR/2017/REC-annotation-vocab-20170223/.

Zhang, H., Monroy-Hernndez, A., Shaw, A., Munson, S. A., Gerber, E., Hill, B. M., Kinnaird, P., Farnham, S. D., Minder, P., and Farnham, Ph.D., S. D. (2014). WeDo: End-To-End Computer Supported Collective Action. In *International AAAI Conference on Web and Social Media*. AAAI.

Zhang, Z., Webster, P., Uren, V., Varga, A., and Fabio, C. (2012). Automatically Extracting Procedural Knowledge from Instructional Texts using Natural Language Processing. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *International Conference on Language Resources and Evaluation (LREC 2012)*, pages 520–527.