electronics

MDPI

Article

Semantic Web Services Ingestion in a Process Mining Framework

Domenico Redavid [1,*,†] and Stefano Ferilli [2,†]

[1] Department of Economic and Finance, University of Bari Aldo Moro, 70124 Bari, Italy
[2] Department of Computer Science, University of Bari Aldo Moro, 70125 Bari, Italy; stefano.ferilli@uniba.it
* Correspondence: domenico.redavid1@uniba.it
† These authors contributed equally to this work.

Abstract: Process mining can be applied to systems for the management of Workflow, Business Processes and, in general, Process-Aware Information to discover and analyse implicit processes. In recent times, semantic interoperability has also become of crucial importance in the area of business processes. In particular, interoperability enables the discovery of new knowledge about processes by exploiting automatic reasoning on information originating from external formal descriptions. To this end, the use of Semantic Web technologies could be one possible solution. Given the different paradigms underpinning the two fields of research, adaptations are needed to realise this solution. In this paper, a possible mapping between Inductive Logic Programming and Semantic Web rules is proposed to discover additional knowledge that can be integrated into the process mining techniques outcomes.

Keywords: semantic web; process mining; SWRL compositions

check for updates

Citation: Redavid, D.; Ferilli, S. Semantic Web Services Ingestion in a Process Mining Framework. *Electronics* **2023**, *12*, 4767. https://doi.org/10.3390/electronics12234767

Academic Editor: Arne Bröring

Received: 6 September 2023
Revised: 2 November 2023
Accepted: 21 November 2023
Published: 24 November 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Business process management (BPM) is the discipline that integrates knowledge from information technology with knowledge from the management sciences, making it applicable to operational business processes [1,2].

As explained in [3], BPM can be interpreted as an extension of Workflow Management (WFM), since WFM focuses primarily on business process automation, while BPM has a broader scope. In fact, BPM covers activities related to process automation and analysis, as well as work management and organisation operations. On one hand, BPM aims to improve operational business processes (e.g., modelling a business process through simulation in order to improve costs and activities). On the other hand, BPM is often associated with software for managing, controlling, and supporting operational processes. Although the latter was the initial goal of WFM, traditional WFM business process automation technology did not consider human factors, offering little managerial support. In addition, there are also so-called process-aware information systems (PAIS) that include WFM/BPM, and systems that offer more flexibility or support-specific processes [4] (such as enterprise resource planning, customer relationship management, case management systems, and rule-based systems). They are considered process-aware, although they do not necessarily control processes through a workflow engine. However, BPM techniques can be applied both to WFM/BPM systems as well as to any PAIS.

In particular, BPM techniques such as process mining [5] are used to discover and analyse implicit processes supported by the systems that manage them. State-of-the-art statistical approaches based on deep learning, such as that described by [6], show some weaknesses in specifying the discovered process, since explainable Artificial Intelligence techniques must be applied in order to describe the process in a readable way. Therefore, these methods are more suitable for problems involving a process-specific activity, as described in [7,8].

In all cases, however, there is a need for a formal representation of information concerning the management and work organisation operations inherent in the processes

that must be interoperable in order to enable automatisation in the current interconnected world of the process manager tasks. The Semantic Web formalisms allow a machine-understandable formal meaning to be associated with the operational and non-operational information, and this process could be the solution to this issue.

There have been some proposals suggested regarding using more abstract standard formalisms to describe processes through Web Ontology Language (OWL) [9] ontologies, i.e., Business Process Management Notation (BPMN) [10], to apply Semantic Web reasoners to accomplish the necessary operations. However, it is not always possible to obtain a complete representation of BPMN artefacts remaining in the decidable fragment of OWL that guarantees the applicability of reasoners [11]. This problem stems from the fact that, historically, process modelling was achieved through formalisms relying on the closed-world assumption, whereas in the Semantic Web, the open world assumption is natively valid. Therefore, representations that might forecast a process model closer to this view, allowing for the advantages of reasoning in Description Logic [12], have been proposed.

The OWL for services (OWL-S) process model, used in this work to map the processes described in *Prolog* generated by a process mining tool, is one such model. This work aims to build a bridge between the formalisms characteristic of Horn clause logic, adopted by Prolog and used in the Workflow Management (WoMan) framework [13], and the formalisms typical of the Semantic Web—in this case, OWL-S and Semantic Web Rule Language (SWRL). In this way, it will be possible to simply and more naturally handle the task of composing Semantic Web Services within more complex workflows processed by WoMan.

Actually, there are some limitations in the representation of processes in OWL, and consequently in OWL-S and SWRL, mainly because to keep the language decidable, certain restrictions are imposed limiting what can be mapped on OWL from FOL. For example, a class cannot be considered an OWL individual (i.e., an instance) of another OWL class at the same time, and meta-modelling is not permitted. Thus, OWL constructs cannot be augmented or redefined. This is reflected in the impossibility of evaluating the service effects without its execution. This leads to the requirement that there must be instances necessary for execution to be able to say that a service or composition of services is valid, and thus also be able to evaluate the effects. In addition, the OWL-S primitives for specifying processes provide for the Iterative construct, which is very limited; however, declarative languages are not allowed to handle an iteration cycle counter directly. Therefore, a complete mapping between the processes described in the WoMan formalism and those of OWL-S is not possible.

The paper is structured as follows: Section 2 reports the related works and Section 3 describes the operational context: details about WoMan, including what predicates are used to describe a workflow, the structure of an OWL-S service and the SWRL syntax, with some conversion issues. Section 4 discusses the formal specification of the problem and related details about the Prolog program used for mapping. Section 5 describes the evaluation results and discusses the problems that emerged. Finally, Section 6 briefly draws some conclusions and describes the outlook.

2. Related Works

There are several works combining OWL ontologies with more powerful logic languages such as FOL. For example, ref. [14] proposed a type of conversion by first normalising FOL sentences into a feature-free prenex conjunctive normal form that eliminates minor syntactic differences and then applying a pattern-based approach to identify common OWL axioms. Another example is the one presented in [15], in which a tool called Gavel with a related OWL-specific extension is described. It allows us to build heterogeneous ontologies containing both FOL and OWL axioms supporting reasoning on the integrated model.

More recently, some methods have been proposed that use prolog directly within OWL reasoners. Lopes et al. [16] present an extension of Nova Hybrid Reasoner (NoHR) [17] that is designed to answer queries over hybrid theories composed of an OWL ontology in

Description Logics and a set of non-monotonic rules in Logic Programming. As argued in the paper, there is a need to combine the distinctive features of these two approaches to knowledge representation and reasoning as dictated by real-world applications, but their integration is still theoretically challenging due to their substantial semantic differences. NoHR supports all polynomial OWL profiles, and even beyond, it to be used with real-world ontologies that do not fit within a single OWL2 profile and has an enhanced integration with its rule engine, which provides support for a vast number of standard built-in Prolog predicates that considerably extend its usability.

Another work that goes in the same direction is the one presented in [18]. It proposes the integration of an OWL reasoner in a framework, named TRILL, containing three probabilistic reasoners written in Prolog in order to manage the non-determinism of tableau methods implemented by Semantic Web reasoners.

From the point of view of the importance of semantic process annotations for enabling interoperability, we cite the following works. Ref. [19] shows how the digital transformation plays a strategic role in simplifying relations with citizens and businesses and in the growth of the community and the economy. The requirement of redesigning processes or creating new ones to ensure that a public service responds to the specific needs of different citizens using a semantic approach for BPMN annotation using domain ontologies is presented. These results have been generalised in [20], which presents a semantic annotation tool for BPMN that allows to identify concepts in workflows that exploit domain ontologies and logical rules and to apply inferential engines against them to enforce these rules.

In [21], the authors propose an approach involving the automatic annotation of semantic components directly in the event logs of a Business process. This is achieved by combining the analysis of textual attribute values, based on a state-of-the-art language model, with novel attribute classification and component categorisation techniques. The approach identifies up to eight semantic components per event, revealing information on the actions, business objects, and resources recorded in an event log.

Ref. [22] presents a framework for the design of public service user interfaces that, on the basis of domain ontologies and BPMn extensions, support the modelling of new interactions with Internet of Things and Bot services in the context of public administration processes. A service semantic annotation model can be shared with and reused by other organisations, thus reducing the user interface design and implementation time, and consequently the overall service development time.

Although interest in the application of reasoning to business processes is making a comeback, there is no recent work on frameworks that directly combine SWRL and Prolog. For the sake of completeness, however, we have included several works on implemented frameworks with these characteristics. Considering the latest theoretical results, they provide a comparative basis for avoiding known problems in the development of new solutions, although, unlike our approach, all these works start from the Semantic Web perspective to obtain the Prolog representation.

In [23] is discussed SWORIER (Semantic Web Ontologies and Rules for Interoperability with Efficient Reasoning), a system that responds to queries about ontologies and rules described in OWL and SWRL (or RuleML) by operating a translation to Prolog and applying the logical reasoning model accordingly. The translation is executed using XSLT (Extensible Stylesheet Language Transformations) to which a set of general rules is added; these rules are implemented directly in Prolog to reinforce the semantics of the OWL primitives. Also discussed in this paper are several problems that the authors had to tackle to make a correct transposition, involving negation, the open world assumption, complementary and disjointed classes, and existential quantification.

SWRL-IQ (Semantic Web Rule Language Inference and Query Tool) [24] is a Protégé plugin [25] that allows to create, edit, save, and submit queries (using SWRL's rule body syntax) to an underlying XSB Prolog-based inference engine [26]. In this case, the OWL-Prolog translator uses an intermediate representation for the knowledge base, modelled in first-order logic (FOL). The rationale for this choice stems from a desire to make the

system highly flexible so that it can be connected to different front-end modules (alternative specification and interaction languages) and back-end modules (other reasoning systems different from Prolog).

SweetProlog [27] is a system used to translate an OWL ontology and rules into a Prolog program. It applies a translation of an OWL ontology (described in Description Logic) and related rules (expressed in OWLRuleML) into a set of facts and a set of Prolog rules. Then, reasoning on these facts and rules is performed by a Prolog interpreter.

3. The Context

Process Mining aims to discover, monitor and improve real processes by extracting knowledge from event logs available in today's information systems [28]. To understand process mining, its constituent elements need to be clearly defined. According to [13,29,30], a process can be defined as a set of actions performed by an agent, and a workflow is the formal representation of a process. A case is said to be the execution of a process, conforming to a given workflow, and it is described as a list events, i.e., identifiable and instantaneous actions, associated with definite temporal moments called steps and collected in traces. A task is a generic task, which is often performed in many instances of the same type. Finally, a task is defined as the actual execution of a task by a resource (an agent who completes it).

Three objectives of Process Mining are also considered in [28]:

- Discovery: this is the most important goal; it allows a process model to be learned from event logs, without the use of additional a priori knowledge;
- Conformance: the model (discovered or hand-built) is compared with the event log in order to detect any differences between modelled and actual behaviours;
- Enhancement: here, the goal is to extend and improve the model using information about the actual process recorded in the event logs.

Several formalisms can be used to construct a process model: among them, one of the most popular is Petri nets [31] or a specialisation of it, WF-nets [32] (WorkFlow Nets). Refs. [33,34] propose a review of the main algorithms and solution strategies based on it.

A different approach is taken by the Declarative Process Mining [35], which represents the model as a set of constraints rather than in a monolithic form (usually through a graph). The *WoMan* framework stands at the intersection of Declarative Process Mining and Inductive Logic Programming (ILP) [36,37]. WoMan pervasively uses First-Order Logic (FOL) as a representation formalism, which guarantees great expressive power and allows context information to be specified through the use of relations [30].

In general, the concept of Process fits well with the context of the Semantic Web [38], in which actions can be interpreted as web services [39] that have been semantically annotated, and a workflow as the composition of multiple services that fulfil a precise material or informational need. This requires, on the one hand, the use of languages and techniques to describe the properties of a web service to enable a machine to understand them, and on the other hand, a way to specify rules for composing services to achieve a given goal. In both cases, there are several frameworks and languages available. In this paper, we will focus on the wide use of OWL-S [40] to represent web service semantics, and on SWRL (Semantic Web Rule Language) [41] to render these semantics in the form of rules. By applying the method presented in [42], we obtain Web services compositions described according to the OWL-S specification by developing an action plan through a backward search algorithm. This action plan could be rendered as an OWL-S composite service, applying the transformations proposed in [43].

It is important to underline the difference in the genesis of the workflows in the two approaches: Process Mining finds the processes through the observation of the logs that are generated during activities execution. On the contrary, Semantic Web Services describe workflows of existing Web services already described semantically through a formal language. In the first case, we can discover processes that include any generic activity, while in the second case, we have services designed for the web that act as

activities that transform one or more inputs into an output. In this work, the objective of mapping is to apply solutions designed for the Semantic Web to enhance process mining techniques. For this reason, there is no correspondence with existing Web services.

Alongside the canonical goals of process mining, in some contexts (e.g., Ambient Intelligence, Industry 4.0), the task of activity prediction could be extremely important. It can be defined as follows: *given a process model and the current (partial) state of the execution of a new process, predict what will be the next activity during execution* [13]. The WoMan framework has been updated to allow for effective activity prediction and, appropriately combined with the Web services composition strategy exposed in [42], has been used to develop a multi-agent architecture for controlling and supervising processes in smart environments [44].

In this section, the two main formalisms of this study are presented: on the one hand, we have WoMan, which takes up the Prolog syntax by introducing some predicates essential to its operations; on the other hand, we have the Semantic Web, and especially the OWL-S and SWRL languages. At the end, some conversion issues are outlined.

3.1. The WoMan Framework

WoMan is a Declarative Process Mining system that automatically and incrementally learns process models. Its high expression power, guaranteed by FOL representations, allows it to represent and manage highly complex cases that cannot normally be handled by classical systems (e.g., those based on Petri Nets). Furthermore, the incremental approach allows it both to learn from scratch and to converge towards correct models using very few example runs, and to manage the context in which the activities take place, thus allowing it to dynamically learn complex pre- and post-conditions, combining the temporal dimension with any other dimension of interest.

In FOL, a predicate, denoted as p/n, expresses a property or relation p on n objects (called its arguments). An atom is a p/n predicate applied to n objects $t_1, \ldots, t_n$, written as $p(t_1, \ldots, t_n)$ indicates that the property or relation p holds for those objects. In Logic Programming, the classical representation states that a comma between two atoms denotes their conjunction, and the implications can be expressed in the form $l_0 : -l_1, \ldots, l_m$, where l_0 is the conclusion and $l_1, \ldots, l_m$ is the conjunction of the premises (to be read as l_0 *is true, if* l_1 *and* ... *and* l_m *are all true*).

Recalling the meaning of trace, event and task as defined in the Section 3, the detailed description of formalisms given in [13] is resumed. In WoMan, the elements of a trace are represented with the following seven tuples denoted as entry:

$$entry(T, E, W, P, A, O, R)$$

where:

1. T is the unique *timestamp* of the event;
2. E is the type of the event (begin_process, end_process, begin_activity, end_activity or context_description);
3. W is the name of the workflow to which the process refers;
4. P is the unique identifier for each process execution;
5. A is the name of the activity;
6. O is the sequence number of occurrence of the activity in the process;
7. R specifies the agent who completes activity A (optional).

Listing 1 shows two traces (where P is 2 and 3) referring to a hypothetical *Case* of an afternoon workflow in a home environment.

Listing 1. Example of a Process *Case* whit two *traces*.

```
...
entry(34,begin_of_process,afternoon,2,start,0).
entry(35,begin_of_activity,afternoon,2,housekeeping,1).
entry(36,end_of_activity,afternoon,2,housekeeping,1).
entry(37,begin_of_activity,afternoon,2,relax,1).
entry(38,end_of_activity,afternoon,2,relax,1).
entry(39,begin_of_activity,afternoon,2,toilet,1).
entry(40,end_of_activity,afternoon,2,toilet,1).
entry(41,begin_of_activity,afternoon,2,dress,1).
entry(42,end_of_activity,afternoon,2,dress,1).
entry(43,begin_of_activity,afternoon,2,out,1).
entry(44,end_of_activity,afternoon,2,out,1).
entry(46,end_of_process,afternoon,2,stop,1).
entry(47,begin_of_process,afternoon,3,start,0).
entry(48,begin_of_activity,afternoon,3,housekeeping,1).
entry(49,end_of_activity,afternoon,3,housekeeping,1).
entry(50,begin_of_activity,afternoon,3,toilet,1).
entry(51,end_of_activity,afternoon,3,toilet,1).
entry(52,begin_of_activity,afternoon,3,relax,1).
entry(53,end_of_activity,afternoon,3,relax,1).
entry(54,begin_of_activity,afternoon,3,coffee,1).
entry(55,end_of_activity,afternoon,3,coffee,1).
entry(56,begin_of_activity,afternoon,3,dress,1).
entry(57,end_of_activity,afternoon,3,dress,1).
entry(58,begin_of_activity,afternoon,3,out,1).
entry(59,end_of_activity,afternoon,3,out,1).
entry(61,end_of_process,afternoon,3,stop,1).
...
```

Given a set of training instances C, WoMan learns the model as a set of atoms and predicates. The core of the model is represented by the following predicates:

- $task(t, C_t)$ — the task t is present in the C_t training cases;
- $transition(I, O, t, C_t)$ — the t transition, present in C_t training cases, is enabled if all input tasks in I are active; if executed (fired), the execution of all tasks in I (in any order) is stopped and the execution of all tasks in O (again, in any order) is started. I and O can be a multi-set.

A limit on the number of possible combinations among transitions is expressed using the following predicate:

- $transition\_provider([\tau_1, \ldots, \tau_n], t, q)$: a transition t, comprising the input tasks in I, is activated if each task in I is produced as the output of a transition τ_k, where τ_k are placeholders (variables) to be interpreted according to the assumption that *"terms denoted by different symbols must be distinct"*. Multiple combinations are allowed, numbered with a progressive value of q, from the cases in C_{tq}.

Other constraints concern the agents that can complete a task. The following predicates fulfil this function:

- $task\_agent(t, A)$: an agent with role A, can perform the task t;
- $transition\_agent([A'_1 \ldots A'_n], [A''_1 \ldots A''_m], t, C_{tq}, q)$: the transition t, which includes input tasks in I and output tasks in O, can take place if each task in I is completed by an agent with role A'_k and each task in O is completed by an agent with role A''_j. Multiple combinations are allowed; they are numbered with a sequential value q, starting with the cases C_{tq}.

Regarding time constraints, there are two possibilities in WoMan: the first is to consider the *timestamp* of events, and the second is the *step*, a unique integer identifier assigned internally by WoMan to each activity. The predicates of interest are as follows:

- $task\_time(t, [b', b''], [e', e''], d)$: the task t must begin at time $i_b \in [b', b'']$ and end at time $i_e \in [e', e'']$, with an average duration equal to d;

- $transition\_time(t, [b', b''], [e', e''], g, d)$: the transition t must begin at time $i_b \in [b', b'']$ and end at time $i_e \in [e', e'']$, with an average duration equal to d (from the beginning of the first task in I to the end of the last task in O), and requires an average time gap equal to g, between the end of the last task in I and the activation of the first task in O;
- $task\_in\_transition\_time(t, p, [b', b''], [e', e''], d)$: the task t, when executed in the transition p, must begin at time $i_b \in [b', b'']$ and must end at time $i_e \in [e', e'']$, with an average duration equal to d;
- $task\_step(t, [b', b''], [e', e''], d)$: the task t must begin at step $s_b \in [b', b'']$ and must end at step $s_e \in [e', e'']$, with an average number of steps equal to d;
- $transition\_step(t, [b', b''], [e', e''], g, d)$: the transition t must begin at step $s_b \in [b', b'']$ and end at step $s_e \in [e', e'']$, with an average number of steps equal to d (from the step of the first task in I to the step of the last task in O) and requires an average gap equal to g steps between the end of the last task in I and the activation of the first task in O;
- $task\_in\_transition\_step(t, p, [b', b''], [e', e''], d)$: the task t, when executed in the transition p, must begin at step $s_b \in [b', b'']$ and end at step $s_e \in [e', e'']$, with an average number of steps equal to d;

where i_b, b', b'', i_e, e', and e'' are relative to the start of the process execution, i.e., they are computed as the timestamp difference between the begin_process event and the event they refer to.

Finally, WoMan can express pre-conditions and post-conditions about the tasks (in general), transitions, and the tasks in the context of a given transition. The major difference between pre- and post-conditions is that in the former case, we refer only to the steps up to the current state, while in the latter case, they can refer to any step, either before or after the current one.

- $act\_T(A, S, R) :- \ldots$ that is, an activity A, of type T can be executed by an agent R, at the step S of executing a case if the body of the rule is satisfied;
- $trans\_P(S) :- \ldots$ a transition P can be executed at the S step of the execution of a case if the body of the rule is satisfied;
- $act\_T\_in\_trans\_P(A, S, R) :- \ldots$ i.e., an activity A, of type T can be executed by an agent R, in the context of a transition P at the step S of executing a case if the body of the rule is satisfied;

where the premises '…' are conjunctions of atoms based on contextual and control flow information.

Conditions refer not only to the current state of execution but can include the state of several *steps*, using the two predicates:

- $activity(s, t)$: at step s, the task t is executed.
- $after(s', s'', [n', n''], [m', m''])$: step s'' follows step s' after a number of steps between n' and n'' and after a time between m' and m''.

After various transformations of the examined cases, the final representation of the learned workflow is obtained. In addition to the list of tasks, this also contains the list of transitions that are the starting point of this work. An example of how they are concretely serialized is the one considered for the evaluation in Section 5.

3.2. The Semantic Web

The Semantic Web [38] was born as an evolution of the World Wide Web to provide a general framework to facilitate machine-oriented sharing and reuse of Web content. The Semantic Web Stack illustrates the architecture of the Semantic Web, highlighting the roles and relationships of the various component elements. The proposed architecture is a source of debate among researchers, as it requires solutions to theoretical problems such as the trade-off between Open vs. closed World Assumption. A variant to the original stack (depicted in Figure 1) proposed in the early years of the Semantic Web is the one described in [45]. The most relevant components that have been developed in the Semantic Web context can be summarised as follows:

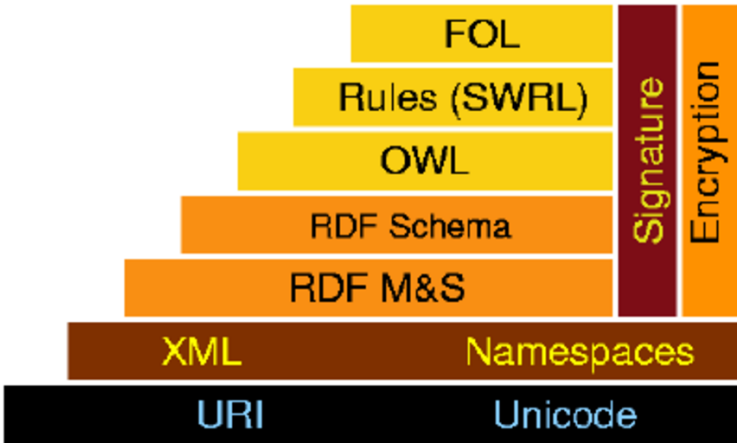

Figure 1. The Semantic Web stack [45].

- Uniform Resource Identifier (URI): provides the means to uniquely identify Semantic Web resources. It has been generalised by the Internationalized Resource Identifier (IRI), which extends the set of characters allowed in resource address specifications.
- eXtensible Markup Language (XML): provides the elementary syntax for structuring content within documents. It does not yet associate any kind of semantics with content.
- Resource Description Framework (RDF): is a very simple language for describing web resources and relationships in the form of subject-predicate-object triples.
- RDF Schema: extends RDF and its vocabulary, adding a semantic layer for hierarchies formed from RDF classes and properties.
- Web Ontology Language (OWL): this is a family of languages suitable for representing knowledge in ontological terms. It augments the vocabulary of the RDF substrate with new properties and classes: for example, relations between classes, the concept of cardinality, equality, characteristics of properties (e.g., symmetry), and enumerative classes are introduced. From OWL, several dialects and derivations (e.g., OWL-Lite, OWL-DL, and OWL-Full) are derived, with goals more focused on their context of use. In 2012, OWL 2 was released; it extended OWL with a small but useful set of features for which effective reasoning algorithms are now available, and those OWL tool developers are willing to support them. The new features include extra syntactic sugar, additional property and qualified cardinality constructors, extended datatype support, simple meta-modelling, and extended annotations. In any case, the trade-off remains among decidability and not fragments.
- Semantic Web Rule Language (SWRL): enriches OWL by allowing knowledge representation in the form of Horn-like clause-based rules.
- First-Order Logic: the abstract layer makes it possible to verify the trustworthiness of the information.

For the purposes of this paper, OWL-S and SWRL will be detailed in greater depth.

3.3. OWL for Services (OWL-S)

OWL-S is an OWL ontology for the description of Semantic Web Services that allows the declarative specification of the semantics of Web services syntactically described with Web Service Description Language (WSDL) [39]. This ontology is designed to describe Web services from three points of view identified by three subclasses:

- Service Profile. It describes the functionality of a service for which it has an advertisement purpose. It is commonly used for the discovery of web services, as it provides different types of information. such as a list of functionalities, a list of service parameters, and a list of service categories.
- Service Model. It specifies how to use the service by describing the semantic content of the possible requests, the conditions to be fulfilled to obtain certain results and, if the services are complex, the process leading to those results. It is possible to define three

types of Process: Atomic, a non-decomposable service that takes a request message and returns a message in response, Composite, which consists of a set of processes specified by a control structure that defines a workflow, and Simple, which represents the abstraction of a compound process that allows it to be seen as an atomic process.
- Service Grounding. This specifies the link from the semantic to the concrete (WSDL) description of the elements that are needed to interact with the service. Thus, the communication protocol, the format in which the message is written, and other useful details are indicated.

Since this work concerns the automatic generation of workflow, as well as more general processes, some details about the OWL-S process model are needed.

The IOPR (Inputs Outputs Preconditions Result) model is the basis of the OWL-S process. Inputs are the information required for the execution of the process; Outputs are the results that the process returns to the requester. The *Preconditions* are restrictions on the *Inputs* that must be valid for the execution of the process. Since different results can be obtained, it is possible to specify conditions, called *inCondition*, that specify when a certain result occurs. The execution of the process produces an *Result* that can be of two types: an *output* or an *Effect* ((intended as a change to the knowledge base). The *Preconditions*, *inConditions* and *Effects*) can be represented with any logical rule language. One way is to represent them as SWRL logical formulae. Finally, the OWL-S process model allows a workflow to be specified using the following control constructs: Sequence, Split, Split-Join, Any-Order, Choice, If-Then-Else, Iterate, Repeat-While and Repeat-Until, and AsProcess.

3.4. Semantic Web Rule Language (SWRL)

SWRL is defined by combining the OWL DL and OWL Lite languages, based on OWL (Web Ontology Language), with a subset of the Rule ML (*Rule Markup Language*). It extends the axioms and classes of OWL by introducing rule definition through Horn-like clauses. An OWL ontology essentially consists of a sequence of axioms and facts. Axioms can be of different natures and the main ones are mentioned here:

- Class: defines a group of individuals that have certain properties in common. For example, Michael and Joan are both members of the class Person. Classes can be organised into specialisation hierarchies using the *subClassOf* axiom. In addition, there is a predefined class called *Thing* that is the superclass of all others, while its counterpart *Nothing* is a subclass of every OWL class.
- Property: defines a relationship between two individuals (*owl:ObjectProperty*) or between an individual and a datum (*owl:DatatypeProperty*). Examples of the first case include properties such as *hasChild*, *hasRelative* or *hasSibling* that establish kinship relationships between people (individuals). An example of the second case might be the hasAge property that validates the age (numeric datum) of a single person (individual). In addition, the domain of a property and its value range can be specified: the former limits the individuals to which the property can be applied, while the latter identifies the set of individuals that the property can have as its value.
- Individual: is an instance of a class or an object of a property. For example, an individual *Francesca* can be described as an instance of the class *Person* and the *hasEmployer* property can be used to bind *Francesca* to the individual *StanfordUniversity* itself an instance of a more generic class University.

The core of the SWRL proposal is the addition of the rule construct as a logical formula to the set of axioms available in OWL (using EBNF notation):

```
axiom ::= rule
```

A rule consists of an antecedent (body) and a consequent (head), each of which consists in turn of a set of atoms (possibly empty). A rule can also be assigned a URI to uniquely identify it. In the abstract syntax, this would correspond to the definition:

```
rule         ::=  'Implies(' [ URIreference ] annotation antecedent consequent ')'
antecedent   ::=  'Antecedent(' atom ')'
consequent   ::=  'Consequent(' atom ')'
```

Informally, a rule can be interpreted in this way: *if the antecedent is true, then the consequent must be also true*. An empty antecedent is always true and an empty consequent is always considered false. Antecedent and consequent are built by conjunctions of atoms: an antecedent (or consequent) is considered true if all the component atoms are true. A single atom can be described abstractly as:

```
atom         ::=  description '(' i-obj ')' | dataRange '(' d-obj ')' |
                  individualvaluedPropertyID '(' i-obj i-obj ')' |
                  datavaluedPropertyID '(' i-obj d-obj ')' |
                  sameAs '(' i-obj i-obj ')' | differentFrom '(' i-obj i-obj ')' |
                  builtIn '(' builtinID d-obj ')'
builtinID    ::=  URIreference
```

In more detail, the atoms can be in the form *C(x)*, *P(x,y)*, *sameAs(x,y)*, *differentFrom(x,y)* or *builtIn(r,x,...)* where *C* is an OWL description or primitive data type, *P* is an OWL property and *r* is a built-in relationship; *x* and *y* can instead be variables, OWL individuals, or OWL primitive data, as appropriate. Informally, an atom *C(x)* is true if *x* is an instance of the class or data range *C*; an atom *P(x,y)* is true if *x* is related to *y* by the property *P*; a *sameAs(x, y)* atom is true if *x* and *y* correspond to the same object; a *differentFrom(x,y)* atom is true if *x* and *y* are interpretable as different objects; finally, *builtin(r, x, ...)* is true if the default relation *r* is true based on the interpretation of its arguments.

SWRL has two different types of concrete syntax, one based on XML and one based on RDF [41]. For the purpose of this paper, we adopt the RDF version as the reference syntax. For example, if we want to represent the following logical rule with RDF syntax:

$$\texttt{hasParent(?x1,?x2)} \wedge \texttt{hasBrother(?x2,?x3)} \Rightarrow \texttt{hasUncle(?x1,?x3)}$$

we obtain the result reported in Listing 2.

Listing 2. Example of SWRL rule representation in RDF syntax.

```
<swrl:Variable rdf:ID="x1"/>
<swrl:Variable rdf:ID="x2"/>
<swrl:Variable rdf:ID="x3"/>
<swrl:Imp>
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasParent"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x2" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasSibling"/>
      <swrl:argument1 rdf:resource="#x2" />
      <swrl:argument2 rdf:resource="#x3" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasSex"/>
      <swrl:argument1 rdf:resource="#x3" />
      <swrl:argument2 rdf:resource="#male" />
    </swrl:IndividualPropertyAtom>
  </swrl:body>
  <swrl:head rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasUncle"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x3" />
    </swrl:IndividualPropertyAtom>
  </swrl:head>
</swrl:Imp>
```

The abbreviation *uri:* is intended only to aid readability. It can be considered as a URI address of a truly defined OWL ontology. Clearly, there is a need for a mechanism to associate a URI with a Prolog artefact in such a way as to ensure the Unique Name Assumption required within the SW.

3.5. Composing OWL-S Process Encoded as SWRL Rules

In this section, we explain how to transform OWL-S process descriptions into a set of SWRL rules to apply a rule-based composition algorithm capable of discovering possible combinations of services.

Since the body and head of the SWRL rule are logical formulae, OWL-S conditions will have to be identified using part of the body or head. Considering that these conditions are expressed on service's *Inputs* and *Outputs*, if the above requirement is met, the conditions will also be expressed in terms of the domain ontology, and thus with the right level of abstraction. After these considerations, we can summarise the procedure that is applied to encode an OWL-S process in SWRL.

- For each result of the process, there is an *inCondition* that expresses the link between the input and the result. This *inCondition* will appear in the body of each resulting rule, while the *Result* will appear in the head. An *inCondition* is valid if it contains all the variables that appear in the *Result*.
- If the result contains an effect composed of several atoms, the rule will be split into as many rules as there are atoms with an inCondition in the body and a single atom in the head.
- The *PreConditions*, since they involve only the *Inputs*, will appear in the body of each resulting rule together with the *inConditions*.

Since it is not necessary for OWL-S atomic services to specify the *inCondition*, we create an ad hoc one that makes the implicit link between inputs and results explicit. For example, let us take a service that, given the name, returns the ISBN of the book. The process will be defined by an input (*?process:BookName*), an output (*?process:ISBN*), and no condition. The resulting rule would be *kb:BookTitle(?process:BookName)* $\rightarrow$*bibtex:Book(?process:ISBN)*, but since the variable *process:ISBN* does not appear in the body of the rule, it would be invalid. We can therefore add in the body of the rule a predicate *hasTransf*, which is always true, that binds each input to the output, thus obtaining a valid rule. In cases where the OWL classes, properties and individuals that form the SWRL rule are defined in different ontologies, it is possible to apply ontology alignment methods [46] that produce OWL class equivalence assertions and add them to the knowledge base containing the SWRL rules.

For our purposes, it is important to emphasise that the SWRL rules under consideration respect two features of the language: each rule must fulfil the *safety* condition, and each rule with a subjunctive consequent is transformed into multiple rules, each with an atomic consequent, by applying Lloyd-Topor [47] transformations. In addition, we impose that a service can be performed only if there are individuals of the knowledge base that make the generated rule grounded, i.e., only SWRL DL-safe rules [48] are considered.

Once we have obtained the SWRL rules representing the services, we can apply composition based merely on Semantic Web languages to obtain the possible combinations of services.

The algorithm behind the *Composer* can be summarised in two basic steps: firstly, the software transforms the OWL-S descriptions of a set of web services into corresponding SWRL rules; subsequently, it uses the newly generated set of SWRL rules as input for an *backward search* algorithm to produce all possible compositions. As a further possibility, the software allows a goal, expressed as an SWRL atom, to be specified so that all compositions are oriented towards satisfying the goal.

With reference to the automatic composition example shown in [42], the scope is to automatically combine the following OWL-S service descriptions: *BookFinder, BNPrice, AmazonPrice and CurrencyConverter*. Suppose you want to know the price of a certain book; for example *'Computers and Brains'* by John von Neumann. For this purpose, the *Composer*

will use the *BookFinder* service to find detailed information based on the title of the book, compose it with the services *BNPrice* and *AmazonPrice* to obtain the price, and finally exploit *CurrencyConverter* to change the currency of the result as needed.

The activities in WoMan workflows are typically generic, but nothing prevents them from concealing calls to remote web services. Since the *Composer* works operationally with SWRL, the aim of this work was aimed, from the outset, at a direct Prolog-SWRL conversion; however, it did not rule out the concrete possibility of returning to OWL-S as a composite service [43].

3.6. Restrictions in the Use of SWRL

In certain contexts, it may be useful to limit the form or expressiveness of rules written in SWRL to increase interoperability, reusability, extensibility, computational scalability, or simplicity of implementation [41].

A first guideline suggests using only classes already defined in OWL (in the same ontology or an external one) in the antecedent and consequent of a rule, preferring them to contextual definitions written directly in the SWRL rule. By following this simple guideline, it becomes easier to translate rules to or from systems including *Prolog*, SQL, systems with production rules (descendants of OPS5) and event–condition–action rules.

A more significant restriction would be to avoid the use of elements such as *owl:someValuesFrom* and *owl:minCardinality* in the consequent of a rule, or *owl:allValuesFrom* and *owl:maxCardinality* in the antecedent of a rule. Both cases are not expressible in Description Logic Programs, as they would correspond to an existential quantification. It is recalled that only universally quantified variables are considered in Horn clauses.

A further restriction concerns the requirement that only variables that appear in the antecedent of the rule may appear in the consequent of the rule (*safety condition*). In our case, this restriction is considered during the mapping of the OWL-S service by adding one or more dummy properties, which are always true, that bind the consequent variables with antecedent ones.

Regarding the inherent limitations of *SWRL* concerning Prolog, we recall that SWRL only supports unary and binary predicates, thus preventing those with arity greater than two, which are regularly allowed in Prolog. In this case, we recall that rules with conjunctive consequents could easily be transformed (via Lloyd–Topor transformations) into multiple rules, each with an atomic consequent. On the other hand, a name can be assigned to a SWRL rule using a URI as a unique identifier; these features have no counterpart in Prolog.

4. Mapping Specification

In this section, the problem of *mapping* between the two formalisms is described in more detail, as is the implementation of the Prolog programme that actually deals with the transformation.

4.1. Formal Specification of the Problem

As mentioned above, the main task of WoMan is to extract a process description from a certain number of cases. The process description is essentially composed of *task*, *transition* and a series of additional properties and constraints related to them. Amongst these, there is emphasis on the *preconditions* that may concern either the individual task or the individual transition, but also a given task in the context of a transition. It is important to emphasise that WoMan can learn several precondition clauses for a certain task, and these clauses are mutually exclusive (the same applies to the two types of preconditions). In particular, given a transition T (I, O, p, C_p), we wish to generate corresponding SWRL rules where:

- The *head* of each rule is composed of a *ClassAtom* SWRL belonging to an OWL class, representing an output $t_o \in O$ of the transition. In fact, to facilitate service composition, each rule that has a consequence consisting of a conjunction of several elements must be transformed into several rules with an atomic consequent, as many as there are elements

in conjunction in the consequent of the source rule (via Lloyd–Topor transformation, as explained in Section 3.5).

- The *body* of each rule consists of two basic parts: the first contains the *ClassAtom* SWRLs (each belonging to an OWL class) representing the input elements $t_i \in I$ of the transition; the second includes the OWL properties that refer to the different preconditions related to the task t_o, the transition p and the task t_o in the context of the transition p. In addition, the body also includes a fixed OWL property named *hasTransf*, which binds the first task t_i to the output t_o, to respect the *safety condition*.

The entire procedure is reported in Algorithm 1.

Algorithm 1 Mapping SWRL in WoMan

for all $transition(I, O, p, C_P) \in W$ **do**
  for all $task(t_O, C_t) \in O$ **do**
    for all $combination C_o p \in C$ Generate a rule S such that **do**
      $S_{body} = map(t_i)|t_i \in I \cup map(c_o p) \cup map(hasTransf(t_i, t_o))|t_i \in I, t_o \in O$
      $S_{head} = map(t_o)$
    end for
  end for
end for

The function *map()* takes care of transforming a WoMan element into its corresponding one in SWRL syntax. A generic task within the input or output of a transition is translated using a *ClassAtom* tag. For example, for task(*Sleeping*,C_t) the corresponding translation in SWRL is reported in Figure 2a.

(a)
```
<swrl:Class Atom>
  <swrl:classPredicate
      rdf:resource="uri:Sleeping"/>
  <swrl:argument1
      rdf:resource="#sleeping"/>
</swrl:Class Atom>
```

(b)
```
<swrl:Class Atom>
  <swrl:classPredicate
      rdf:resource="uri:Predicate"/>
  <swrl:argument1
      rdf:resource="#x"/>
</swrl:Class Atom>
```

(c)
```
<swrl:Individual Property Atom>
  <swrl:propertyPredicate
      rdf:resource="uri:Predicate"/>
  <swrl:argument1
      rdf:resource="#predicate"/>
  <swrl:argument2
      rdf:resource="#x"/>
</swrl:IndividualPropertyAtom>
```

(d)
```
<swrl:Individual Property Atom>
  <swrl:propertyPredicate
      rdf:resource="uri:Predicate"/>
  <swrl:argument1
      rdf:resource="#x"/>
  <swrl:argument2
      rdf:resource="#y"/>
</swrl:IndividualPropertyAtom>
```

Figure 2. SWRL Rendering of WoMan elements.

A unary predicate in Prolog of the type $predicate(x)$ can be interpreted in two distinct ways depending on the definition of x: if x represents an instance of the class *predicate* then the correct transformation is with a ClassAtom, as shown in Figure 2b. If x represents something conceptually different from *predicate*, then the most suitable SWRL counterpart is a property such as the one in Figure 2c. For example, the predicate $activity1(X)$ might have been defined, where X represents the *step* or *timestamp* at which the activity *activity1* occurred. Conversely, a predicate of the type $agent(Y)$ might have been defined, such that Y represents an agent. A binary Prolog predicate of the type $predicate(x, y)$ may instead be converted directly into an *IndividualPropertyAtom*, as shown in Figure 2d.

Finally, due to the limitation of SWRL, whereby no atoms have arity greater than two, the n-ary predicates definable in Prolog must be decomposed. In order to work with n-ary, some transformations can be applied [49]. For the sake of simplicity, only mapping with binary atoms is considered in this paper.

Let consider the simple workflow learned by WoMan reported in Listing 3:

Listing 3. An example of workflow learned by WoMan.

```
task(start,[1-1]).
task(housekeeping,[1-3]).
task(prepare_snack,[1-1]).
transition([start]-[housekeeping],p1,[1-1]).
transition([housekeeping]-[prepare_snack],p16,[1-1]).
actpreparesnack(A,B):-activity(C,X,B),agent(B),next(A,C).
```

Applying the Algorithm 1 to the example workflow, we obtain the SWRL rules reported in Listing 4:

Listing 4. SWRL rules obtained from the WoMan workflow of Listing 3.

```
<swrl:Imp rdf:ID="Rulep11">
  <swrl:body parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasTransf"/>
      <swrl:argument1 rdf:resource="#start"/>
      <swrl:argument2 rdf:resource="#housekeeping"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="uri:Start"/>
      <swrl:argument1 rdf:resource="#start"/>
    </swrl:ClassAtom>
  </swrl:body>
  <swrl:head parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="uri:HouseKeeping"/>
      <swrl:argument1 rdf:resource="#housekeeping"/>
    </swrl:ClassAtom>
  </swrl:head>
</swrl:Imp>
<swrl:Imp rdf:ID="Rulep161">
  <swrl:body parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasTransf"/>
      <swrl:argument1 rdf:resource="#housekeeping"/>
      <swrl:argument2 rdf:resource="#preparesnack"/>
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:PreTask"/>
      <swrl:argument1 rdf:resource="#x0"/>
      <swrl:argument2 rdf:resource="#x1"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="uri:Activity"/>
      <swrl:argument1 rdf:resource="#activity"/>
    </swrl:ClassAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasStep"/>
      <swrl:argument1 rdf:resource="#x2"/>
      <swrl:argument2 rdf:resource="#activity"/>
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasActivity"/>
      <swrl:argument1 rdf:resource="#x3"/>
      <swrl:argument2 rdf:resource="#activity"/>
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:hasAgent"/>
      <swrl:argument1 rdf:resource="#x1"/>
      <swrl:argument2 rdf:resource="#activity"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="uri:Agent"/>
      <swrl:argument1 rdf:resource="#x1"/>
    </swrl:ClassAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="uri:Next"/>
      <swrl:argument1 rdf:resource="#x0"/>
      <swrl:argument2 rdf:resource="#x2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="uri:HouseKeeping"/>
      <swrl:argument1 rdf:resource="#housekeeping"/>
    </swrl:ClassAtom>
  </swrl:body>
  <swrl:head parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="uri:PrepareSnack"/>
      <swrl:argument1 rdf:resource="#preparesnack"/>
    </swrl:ClassAtom>
  </swrl:head>
</swrl:Imp>
```

4.2. Using of Ontologies

Before delving into the implementation phase, a premise must be selected: to enable the association of WoMan concepts and relationships with related alter egos in SWRL, it is necessary to identify existing OWL ontologies that semantically reflect Prolog predicates and to create ad hoc OWL ontologies if these are not available.

Therefore, to test the operation of the application on a concrete WoMan model, two OWL ontologies were created using *Protégé ver. 5.5* with OWL support and published via a generic Apache web server: the first, named womanOntology.owl, contains fixed WoMan properties and concepts, independent of any workflow and usage scenario. Specifically, the OWL classes identified are the following (Figure 3):

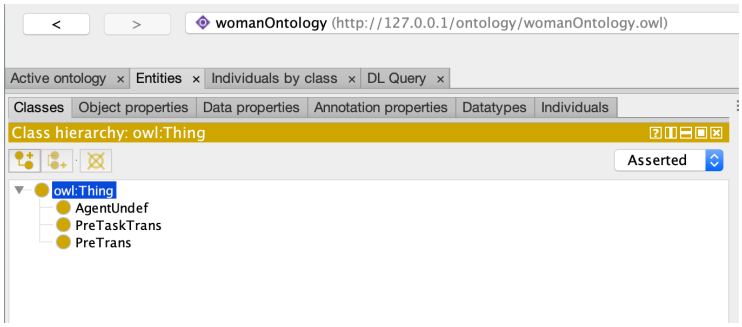

Figure 3. WoMan OWL classes.

- PreTrans: identifies the correspondent of the WoMan predicate *trans_T(S)* concerning preconditions on transitions;
- PreTaskTrans: used in the decomposition of the ternary predicate *act_T_in_trans_P(A,S,R)* to indicate the preconditions related to a task in the context of a transition;
- AgentUndef is a predicate that recurs in preconditions; it indicates that the argument is an agent type that is not precisely defined.

In addition to these, the following properties have been identified and created (Figure 4):

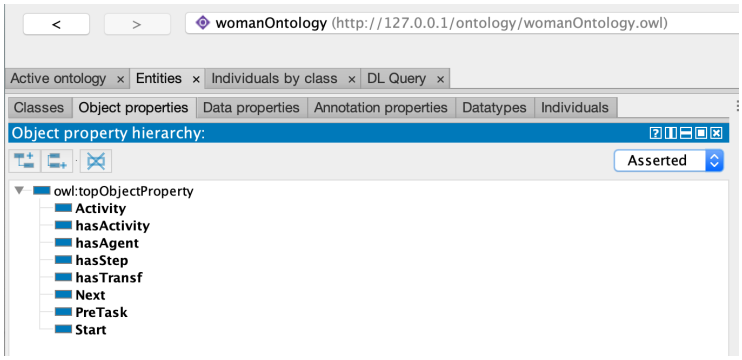

Figure 4. WoMan OWL properties.

- PreTask: corresponds to the predicate used as the head in the preconditions related to a task in WoMan;
- hasActivity: used in the decomposition of *PreTaskTrans* to indicate the relation with the argument A, relating to an activity;
- hasStep: used in the decomposition of *PreTaskTrans* to indicate the relationship with the S argument, relative to the execution step;
- hasAgent: used in the decomposition of *PreTaskTrans* to indicate the relationship with the R argument, relating to the agent performing the task;
- Next: the corresponding concept of the predicate $next(s_1,s_2)$, which specifies the order relation between two distinct execution steps, whereby the first argument precedes the second.

The second ontology, known as *exampleOntology.owl*, contains only OWL classes related to a specific test case used for the evaluation. At the highest level of the hierarchy, the following OWL classes have been created (Figure 5):

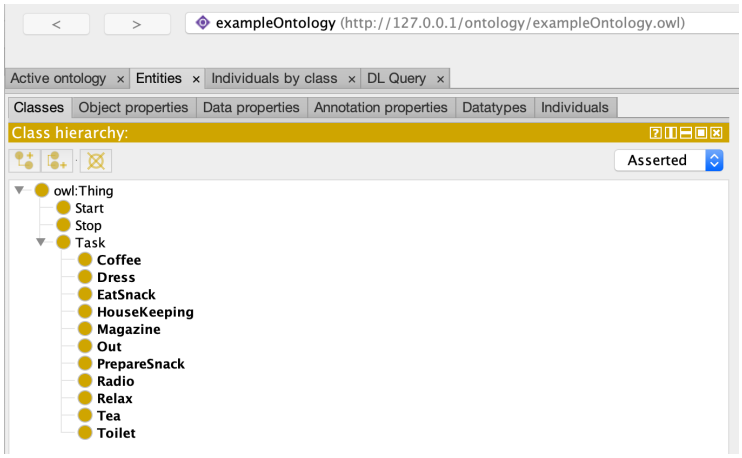

Figure 5. OWL ontology example.

- Start: is the counterpart of the task *start*, which is used as a placeholder to indicate the start of a process;
- Stop: is the counterpart of the task *stop*, which is used as a placeholder to indicate the end of a process;
- Task: is the superclass of any specific task within a workflow or application scenario. Since the test workflow concerns certain tasks typically carried out in-house, subclasses such as *HouseKeeping*, *Tea* and *Dress* have been defined.

The declaration of the properties and characteristics of the various concepts, such as *Property* domain and range, was not elaborated because it was considered unnecessary, so the default values (both for domain and range set to *owl:Thing*) were left. Although not currently needed for the purposes of this paper, these descriptions should be specified according to the reasoning goals to be applied from the Semantic Web side.

4.3. Implementation

The The mapping procedure between a WoMan workflow and the corresponding set of SWRL rules was developed in Prolog and tested with the Yet Another Prolog (YAP) [50] compiler. This section briefly introduces the basic components and specifically discusses on the most relevant predicates.

First of all, the Prolog code is divided into three separate parts:

- Predicates Mapping: represents the core of the procedure, where the main predicates involved in the transformation process reside;
- Ontology Mapping: contains the associations between the names of tasks and predicates used in a workflow and the URIs of the resources of some OWL ontologies;
- Utilities: contains a set of more general utility predicates that are exploited by the *Predicates Mapping* part.

4.3.1. Predicates Mapping

In *Predicates Mapping* procedure the main predicate is *map(Path)* which is responsible for transforming the WoMan workflow specified in the *Path* into a set of SWRL rules stored in an output file. For this purpose, it calls the following predicates in the given order:

- *load_input(Path,VarList):* takes care of reading the input files required for mapping. In particular, *Path* indicates the path to the workflow file from which the other three pre-condition files are derived, i.e., tasks, transitions and tasks in the context of transitions, respectively. All identified variables are stored in *VarList*;

- *print_section_header(Path):* uses the contents of an auxiliary file (named *basicStub.txt*) as the output file header. It represents a standard header for an OWL/SWRL file and must contain the ontology references for all concepts that will appear in the rules. The choice to use an external file instead of implementing the same logic within the code stems from the need to facilitate the modification and addition of new ontologies in the output file header, which is highly dependent on the specific input workflow;
- *print_section_variables(VarList):* writes all previously recognised variables in one macro-block of the output file. As the variables read from the input files are identified by Prolog through its internal representation (e.g., 6789), each variable is appropriately renamed appending a numerical identifier (e.g., $\times 1$, $\times 2$, ... $\times 100$, etc.).
- *print_section_rules:* generates one or more rules for each transition in the knowledge base and writes them to the output file.

Within the print section rules, the predicate that concretely implements the mapping logic for a single transition is called, i.e., *map_ transition(I,O,Id)*: this takes up the arguments that characterise the definition of an transition in WoMan, avoiding the specification of cases as irrelevant for mapping purposes. On the right-hand side of *map_transition* we can distinguish two essential components:

- *print_body(I,O,PreTasks,PreTransitions,PreTaskTransition):* generates the body of a SWRL rule with the tasks in I and the preconditions specified in *PreTask*, *PreTransitions* and *PreTaskTransition*. As explained above, in the case in which several preconditions that can be associated with the same transition, a rule is created for each possible combination. Furthermore, to respect the *safety condition*, it adds a dummy property *hasTransf* that links the first element in *I* with the task in *O*.
- *print_head(O):* generates the head of the SWRL rule, turning the task in *O* into an *ClassAtom* OWL. It is important to note that for each rule, there is only one output element: if a transition has *n* tasks in *O* (n >1), these are decomposed to form *n* rules with identical input and atomic output.

A detailed description of all other minor program predicates and their operation are documented in the source code.

4.3.2. Ontology Mapping

In *Ontology Mapping* the task associations are distinguished from the predicates that appear in the WoMan preconditions by means of the following procedures:

- *task_ontology_map(<name task>, <URI>)*
- *predicate_ontology_map(<predicate name>, [<URI>list])*

In the first case, the association is direct, i.e., the name of a certain task in WoMan corresponds to a reference to a specific concept of some ontology. The second case holds a number of possibilities that are worth investigating: If the predicate we wish to associate is *unary*, the correct specification of the association allows us to resolve the ambiguity in the translation discussed in the previous paragraph. In fact, if the predicate is interpretable as a *ClassAtom*, the list of URIs must consist of only one element: the class reference. Otherwise, the predicate implies a binary property, so the list of URIs must be composed in order of: *reference to the property*, *reference to the class of the first argument*, *reference to the class of the second argument*. Currently, the last two references are not used in the mapping and may also be empty. The list contains more than one item, which is relevant to the distinction.

If the predicate to be mapped is binary, the list of URI must consist of a single element: the reference to the corresponding *IndividualPropertyAtom*.

The choice of the list as the second argument in all predicate types—even when not strictly necessary, as for binary predicates—is motivated, on the one hand by reasons of *coherence* and code cleanliness, and on the other hand by the desire to facilitate subsequent software evolutionary interventions. The flexibility of the list allows the easy insertion of additional elements to better describe a predicate and its characteristics.

5. Evaluation

The functioning of the software programme was verified on the model learned by WoMan from the dataset *Afternoon* (shown in Listing 5). The model contains the log of approximately 1000 process executions. It is characterised by a set of household activities typically carried out in the afternoon, such as household chores, preparing a snack, or enjoying a hot tea.

A thorough quantitative analysis would require much effort from human experts to manually check all the many process cases generated by the compositions to determine why they are not compliant with the original model and whether they make sense or not. For this reason, we will leave evaluation experiments for future work and will focus on a qualitative evaluation on a sample real-world domain in this study. Still, a few quantitative metrics useful for assessing the effectiveness and efficiency performance of our approach can be identified. For efficiency, we may compute the total runtime for a mapping and the average runtime per translated transition. For effectiveness, we may count how many new cases are generated by the SWRL compositions, how many of them do or do not make sense, the compositions that cause them, and associated percentages. Our qualitative evaluation will investigate the mapping process from two different perspectives: on the one hand, we wish to judge its formal correctness and highlight any open problems; on the other hand, we wish to exploit it in conjunction with the web services *Composer* described in [42], to analyse the compositions produced, and to compare them with the model learned from WoMan.

These two perspectives are studied in detail in Sections 5.1 and 5.2.

Listing 5. WoMan-learned model based on the Afternoon dataset. For simplicity, only the model transitions are reported.

```
...
transition([act_magazine,act_radio]-[stop],23).
transition([act_eat_snack]-[act_magazine],24).
transition([act_eat_snack,act_magazine]-[stop],26).
transition([act_radio]-[act_magazine],27).
transition([act_prepare_snack]-[act_eat_snack,act_radio],25).
transition([act_tea]-[act_dress],22).
transition([act_coffee]-[act_dress],15).
transition([act_relax]-[act_toilet],14).
transition([act_toilet]-[act_prepare_snack],16).
transition([act_tea]-[act_toilet],17).
transition([act_out]-[stop],1).
transition([act_dress]-[act_out],2).
transition([act_toilet]-[act_dress],3).
transition([act_coffee]-[act_toilet],4).
transition([act_housekeeping]-[act_relax],6).
transition([act_eat_snack,act_magazine,act_radio]-[stop],8).
transition([act_prepare_snack]-[act_eat_snack,act_magazine,act_radio],9).
transition([act_magazine]-[act_radio],19).
transition([act_prepare_snack]-[act_eat_snack,act_magazine],20).
transition([act_tea]-[act_prepare_snack],10).
transition([act_relax]-[act_tea],11).
transition([act_eat_snack,act_radio]-[stop],18).
transition([act_magazine]-[act_eat_snack],28).
transition([act_prepare_snack]-[act_magazine,act_radio],29).
transition([act_coffee]-[act_prepare_snack],21).
transition([act_relax]-[act_coffee],5).
transition([act_toilet]-[act_relax],12).
transition([act_housekeeping]-[act_toilet],13).
transition([start]-[act_housekeeping],7).
...
```

5.1. Mapping Analysis

The Prolog program in charge of mapping accurately fulfils the requirements outlined in the formal analysis phase of the problem. Moreover, performs its task quickly, with infinitesimal execution times for 36 transitions and 64 preconditions.

However, the execution of a concrete WoMan model revealed some relevant problems to be highlighted.

The first problem concerns the presence of duplicate rules: it is configured as the result of the rule decomposition mechanism having multiple tasks in output. Take, for example, the following two transitions extracted from the model:

- `transition([act_prepare_snack]−[act_eat_snack,act_magazine,act_radio], 9)`
- `transition([act_prepare_snack]−[act_eat_snack,act_magazine], 20)`

In this case, the decomposition of the first transition will produce the three rules below (described with simplified notation):

- `act_eat_snack` ← `act_prepare_snack`
- `act_magazine` ← `act_prepare_snack`
- `act_radio` ← `act_prepare_snack`

and for the second transition, they will be generated:

- `act_eat_snack` ← `act_prepare_snack`
- `act_magazine` ← `act_prepare_snack`

It can be observed that the first two rules are repeated. While the deletion of duplicates would increase the efficiency of the *Composer*—which would not have to repeat the same computation for all duplicates—on the other hand, the direct association with the source transitions would be lost. In the example, if the rules generated by the second transition were deleted, a complete trace of them would be lost in the output file.

A second set of problems concerns the management of preconditions. The addition of all types of preconditions in the body of a SWRL rule makes it more complicated to distinguish which predicates belong to one precondition rather than another. One way around this problem exploits the special concepts *PreTask*, *PreTrans*, *PreTaskTrans*: in fact, all properties and concepts that follow *PreTask*, *PreTrans* and *PreTaskTrans* in the body of the rule will correspond to the specification of a precondition related to the task, the transition, or the task in the context of the transition, respectively. This *workaround* may not be suitable in all circumstances, which is why it may be required to introduce some way to formally associate the body with the precondition reference.

Such reflections could actually be extended to all variables, even those within unary and binary predicates, whereby a different name does not necessarily correspond to an inequality on the semantic level.

5.2. Analysis of the Compositions

As pointed out in the introduction, the world of web services has numerous similarities with that of processes, in which an activity can be seen as a service and a process as a sequence of services linked together to achieve a particular goal. Consequently, there is a strong interest in studying the results of a composition algorithm devised for Web services, but which is instead applied to the domain of Process Mining. The functioning of the *Composer* [42] is conceptually simple: given a set of SWRL rules, for each of them it takes, one at a time, the elements of the body and looks for all possible correspondences with the heads of the other rules; when a correspondence is found, it considers the body of the identified rule and the process repeats, considering all possible paths. Since the rules produced by the mapping process can only have the tasks in the head, all properties and concepts related to the preconditions are not taken into account by the *Composer*; this is the reason why preconditions are not reported.

In Listing 6, we can examine the list of rules used for composing (the property *hasTransf* is abbreviated as *hT*.

Looking at the set of Prolog rules and considering the mechanism of operation of the composer, potential loops that can be triggered between two or more rules were detected. For example, taking the rules 27 and 19:

- 27) hT(act_radio, act_magazine), act_radio → act_magazine
- 19) hT(act_magazine, act_radio), act_magazine → act_radio

Listing 6. List of input rules to the Composer, obtained as a result of mapping. The line number is the rule identifier of the related transition in the model learned from WoMan. SWRL atom variables were omitted for readability.

```
23) act_magazine ∧ act_radio ∧ hT(act_magazine,stop) → stop
24) act_eat_snack ∧ hT(act_eat_snack,act_magazine) → act_magazine
26) act_eat_snack ∧ act_magazine ∧ hT(act_eat_snack,stop) → stop
27) act_radio ∧ hT(act_radio,act_magazine) → act_magazine
25.1) act_prepare_snack ∧ hT(act_prepare_snack,act_eat_snack) → act_eat_snack
25.2) act_prepare_snack ∧ hT(act_prepare_snack,act_radio) → act_radio
22) act_tea ∧ hT(act_tea,act_dress) → act_dress
15) act_coffee ∧ hT(act_coffee,act_dress) → act_dress
14) act_relax ∧ hT(act_relax,act_toilet) → act_toilet
16) act_toilet ∧ hT(act_toilet,act_prepare_snack) → act_prepare_snack
17) act_tea ∧ hT(act_tea,act_toilet) → act_toilet
1) act_out ∧ hT(act_out,stop) → stop
2) act_dress ∧ hT(act_dress,act_out) → act_out
3) act_toilet ∧ hT(act_toilet,act_dress) → act_dress
4) act_coffee ∧ hT(act_coffee,act_toilet) → act_toilet
6) act_housekeeping ∧ hT(act_housekeeping,act_relax) → act_relax
8) act_eat_snack ∧ act_magazine ∧ act_radio ∧ hT(act_eat_snack,stop) → stop
9.1) act_prepare_snack ∧ hT(act_prepare_snack,act_eat_snack) → act_eat_snack
9.2) act_prepare_snack ∧ hT(act_prepare_snack,act_magazine) → act_magazine
9.3) act_prepare_snack ∧ hT(act_prepare_snack,act_radio) → act_radio
19) act_magazine ∧ hT(act_magazine,act_radio) → act_radio
20.1) act_prepare_snack ∧ hT(act_prepare_snack,act_eat_snack) → act_eat_snack
20.2) act_prepare_snack ∧ hT(act_prepare_snack,act_magazine) → act_magazine
10) act_tea ∧ hT(act_tea,act_prepare_snack) → act_prepare_snack
11) act_relax ∧ hT(act_relax,act_tea) → act_tea
18) act_eat_snack ∧ act_radio ∧ hT(act_eat_snack,stop) → stop
28) act_magazine ∧ hT(act_magazine,act_eat_snack) → act_eat_snack
29.1) act_prepare_snack ∧ hT(act_prepare_snack,act_magazine) → act_magazine
29.2) act_prepare_snack ∧ hT(act_prepare_snack,act_radio) → act_radio
21) act_coffee ∧ hT(act_coffee,act_prepare_snack) → act_prepare_snack
5) act_relax ∧ hT(act_relax,act_coffee) → act_coffee
12) act_toilet ∧ hT(act_toilet,act_relax) → act_relax
13) act_housekeeping ∧ hT(act_housekeeping,act_toilet) → act_toilet
7) start ∧ hT(start,act_housekeeping) → act_housekeeping
```

In this case, the *Composer* sets as a target *act_magazine* from rule 27 and picks up the body of it (*act_radio*), finding a *match* with the head of rule 19. At this point, the body of rule 19 is considered and the algorithm looks for all rules that present the task *act_magazine* in its own head. Rule 27 is then chosen once again, causing the the loop to restart indefinitely. The composer tool already manages this issue, but to enhance performances, all problematic Prolog rules are identified and removed from the input: therefore, rules 19, 28 and 12 are not considered.

Beyond the evaluation of the compositions themselves, what is interesting to study here is whether the aforementioned compositions can provide a novel process *cases* concerning the model learned from WoMan, possibly contributing to its enrichment. To this end, each composition that had *Start* as its origin and *Stop* as its termination was converted into a process case (in the syntax described in Section 3.1) and added to a log. The log was then fed to the conformance checking module of WoMan to detect divergent cases from the model. WoMan provides different types of *warnings* that can arise from the comparison between a pre-existing model and a new process case. All *warnings* are reported at the level

of a single event, and there are three types from which it can be inferred whether the case in question represents a sequence of activities never encountered before. In details:

- task: an unknown *task* was detected;
- transition: a new *transition* was discovered;
- history: the *case* differs from the history of processes already analysed.

A practical evaluation was run on an Quad-Core Intel Core i5 (2 GHz) processor with 16 GB of RAM under macOS 10.15.7. Submission of the *log* generated from the compositions resulted in 36 new process cases and, for all of them, the type of *warning* reported was *transition*, with a total runtime for a mapping of 76 milliseconds resulting in a 2.11 milliseconds for transition. From the point of view of effectiveness, 90 new cases were generated from the SWRL compositions: 36 were new cases (i.e., 40 per cent), and 31 out of 36 (i.e., about l86 per cent) make sense considering that the cases were observed in the afternoon but that this domain also includes the morning and evening, when they might also occur.

After further analysis of the results, it was discovered that the compositions of two well-defined rules were responsible for all the new cases, and specifically rules 24 and 27. In the remainder of this section, these cases are explained and commented on. To aid readability, we assume the existence of a single ontology *o* in which all concepts of interest are defined (which would constitute the union of the two ontologies described in Section 4.3).

Listing 7 reports the possible SWRL compositions for rule 24, where the indentation is in line with the rule containing the atom to which the line refers. All compositions have the task *Magazine* as their goal. Thus, matches are sought for the *EatSnack* atom, identifying the *PrepareSnack* → *EatSnack* rule. The latter is repeated identically in 25.1, 9.1 and 20.1, but is considered by Composer only once. At this point, one looks for a *match* with *PrepareSnack* and finds it at rules 16, 10 and 21. Based on 16, one considers the task *toilet* that appears in the head of rules 14, 13, 4 and 17. Starting with 14, the only possible path is in composition with rule 6, ending with rule 7. With 13, we come directly to 7, since it is the only one in which the task *housekeeping* figures as the head. The last two cases are led back to rule 6 in the same way, and consequently to rule 7. Going back to *PrepareSnack*, in the last two scenarios (rules 10 and 21), we arrive at the *Relax* atom with rules 11 and 5, respectively, ending with the binomial *Start* → *HouseKeeping*.

Listing 7. SWRL compositions starting from Rule 24.

```
o:PrepareSnack(?prepare_snack) ∧ o:hT(?prepare_snack, ?eat_snack) → o:EatSnack(?eat_snack)
  o:Toilet(?toilet) ∧ o:hT(?toilet, ?prepare_snack) → o:PrepareSnack(?prepare_snack)
    o:Relax(?relax) ∧ o:hT(?relax, ?toilet) → o:Toilet(?toilet)
      o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
        o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
    o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?toilet) → o:Toilet(?toilet)
      o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
    o:Coffee(?coffee) ∧ o:hT(?coffee, ?toilet) → o:Toilet(?toilet)
      o:Relax(?relax) ∧ o:hT(?relax, ?coffee) → o:Coffee(?coffee)
        o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
          o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
    o:Tea(?tea) ∧ o:hT(?tea, ?toilet) → o:Toilet(?toilet)
      o:Relax(?relax) ∧ o:hT(?relax, ?tea) → o:Tea(?tea)
        o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
          o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
  o:Tea(?tea) ∧ o:hT(?tea, ?prepare_snack) → o:PrepareSnack(?prepare_snack)
    o:Relax(?relax) ∧ o:hT(?relax, ?tea) → o:Tea(?tea)
      o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
        o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
  o:Coffee(?coffee) ∧ o:hT(?coffee, ?prepare_snack) → o:PrepareSnack(?prepare_snack)
    o:Relax(?relax) ∧ o:hT(?relax, ?coffee) → o:Coffee(?coffee)
      o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
        o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
```

For all reported compositions, the task from which the case stood out from the model was precisely the goal *Magazine*. Since the sequence up to the task *EatSnack*—in any reported combination—turned out to be consistent with the model and it seems plausible

to engage in reading a newspaper immediately after eating a snack, it must be inferred that the compositions up to goal *Magazine* are also entirely plausible. As a further confirmation, it is pointed out that there were no previously performed activities that may conflict with *Magazine*, nor was the reading of the newspaper previously performed (it remains possible, however, that a newspaper may be read at two different times in the afternoon or that the object of reading is a different newspaper from time to time).

In rule 27, whose SWRL compositions are reported in Listing 8, the goal *Magazine* leads immediately to the atom *Radio* that appears as the head in the PrepareSnack rule. *Radio* is repeated in the input three times (rules 9.3, 25.2, 29.2) but considered only once. The atom *PrepareSnack* finally generates the paths already seen in the previous cases.

Again, the novelty here is the placement of the goal *Magazine* in the sequence of activities, while the coherence of the flow is preserved. In fact, reading the newspaper immediately after listening to the radio could be motivated by a desire to learn more about a news item heard on some radio station. In any case, even if the two activities were completely disconnected, the sequence would remain plausible. The only anomaly from the case above is that the action *PrepareSnack* is not followed by *EatSnack*, but there is nothing to prevent the snack from being prepared for someone else, such as by a parent for a child.

Listing 8. SWRL compositions starting from Rule 27.

```
o:PrepareSnack(?prepare_snack) ∧ o:hT(?prepare_snack, ?radio) → o:Radio(?radio)
  o:Toilet(?toilet) ∧ o:hT(?toilet, ?prepare_snack) → o:PrepareSnack(?prepare_snack)
    o:Relax(?relax) ∧ o:hT(?relax, ?toilet) → o:Toilet(?toilet)
      o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
        o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
    o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?toilet) → o:Toilet(?toilet)
      o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
    o:Coffee(?coffee) ∧ o:hT(?coffee, ?toilet) → o:Toilet(?toilet)
      o:Relax(?relax) ∧ o:hT(?relax, ?coffee) → o:Coffee(?coffee)
        o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
          o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
    o:Tea(?tea) ∧ o:hT(?tea, ?toilet) → o:Toilet(?toilet)
      o:Relax(?relax) ∧ o:hT(?relax, ?tea) → o:Tea(?tea)
        o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
          o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
  o:Tea(?tea) ∧ o:hT(?tea, ?prepare_snack) → o:PrepareSnack(?prepare_snack)
    o:Relax(?relax) ∧ o:hT(?relax, ?tea) → o:Tea(?tea)
      o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
        o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
  o:Coffee(?coffee) ∧ o:hT(?coffee, ?prepare_snack) → o:PrepareSnack(?prepare_snack)
    o:Relax(?relax) ∧ o:hT(?relax, ?coffee) → o:Coffee(?coffee)
      o:HouseKeeping(?housekeeping) ∧ o:hT(?housekeeping, ?relax) → o:Relax(?relax)
        o:Start(?start) ∧ o:hT(?start, ?housekeeping) → o:HouseKeeping(?housekeeping)
```

Analysing the Rules that have the consequent *Stop* as their goal (i.e., 23, 26, 1, 8, and 18), we observe that they reuse the already obtained compositions except for rule 1 which does not compose with any other rule.

In particular, for rule 23, for each possible composition, the tasks from which the case was distinguished from the model are *Magazine* and *EatSnack* within the body of the starting rule. Similarly, for rule 26, the tasks from which the case was distinguished from the model are *Magazine* and *Radio*, whereas in rule 8, we have three atoms: *EatSnack*, *Magazine* and *Radio*. Finally, rule 18 has, in the body, the conjunction of *EatSnack* and *Radio*. It can easily recognise that for these rules, all the sub-compositions seen for rules 24 and 27 are re-applicable; in particular, paths departing from *Magazine* are the only ones of interest for analysis.

For all other conforming compositions, the genesis is briefly commented on based on the list of rules reported in Listing 6.

According to rule 7, the goal *housekeeping* is set and the algorithm proceeds to examine the body, namely the task *Start*. Since it represents the start signal of a process, it can never appear in the head of a rule; therefore, no compositions are expected.

In all three cases in which the goal is *PrepareSnack*, the compositions are those already covered by most of the rules seen above.

Take the objectives *dress* of rule 3 and *PrepareSnack* of rule 16; the task *toilet* that appears in the head of rules 14, 13, 4 and 17 is considered the next objective. From 14, the only possible path is in composition with rule 6, ending in rule 7. From 13, we come directly to 7, since it is the only one in which the task *housekeeping* appears as the head. The last two cases lead back in the same way to rule 6 and, consequently, to rule 7.

Rules 22, 17 and 10, though they have different goals, have the same body (only the *Tea* atom), and thus share the same set of compositions. Since the atom *Tea* appears as a head only in Rule 11, it represents the only possible composition path.

As in the previous case, rules 4, 15 and 21 have the same atom *Coffee* in their bodies. Again, the atom appears as a head in only one rule: rule 5.

The goal *Out* recalls *Dress* atom in rule 2. *Dress* appears as a head in rules 3, 22 and 15 from which, again, known paths depart.

As can be seen, rules 14, 6, 13, 5 and 11 do not generate interesting compositions, but ones with very limited depth. As can be inferred by looking at the results, all the compositions presented appear correct and consistent with the starting model. Furthermore, the new process cases identified from the compositions appear plausible and represent excellent candidates for extending the WoMan model.

Finally, a note of credit goes to the *Composer* software version 2.1, which, thanks to optimised handling of duplicate rules, makes it possible to maintain a 1:1 mapping with the relevant WoMan model, without decreasing efficiency.

6. Conclusions and Outlook

The relationship between Process Mining and Semantic Web represents a field of research that has only been partially explored. This paper contributes to the direction of possible combinations between the Prolog and SWRL languages, but there are numerous aspects to be further investigated and possible future developments to consider.

A first extension concerns the enrichment of SWRL rules by including WoMan post-conditions. At the current state, only tasks, transitions and preconditions are considered for transformation. In addition, there exist in WoMan some predefined predicates—such as *next* or *after*—in which there is an implicit order relation between the arguments, the semantics of which could be made explicit in SWRL through the use of the *built-in*.

Another aspect that would deserve further investigation concerns the possibility of mapping involving multiple WoMan models. The resulting set of rules could be given as input to the Composer, under the probable assumption that completely new activity plans unknown to the models taken individually would be generated.

Furthermore, there are two aspects related to the single research areas to be tackled: from the WoMan-side in relation to the fact that rarer process components could be eliminated in favour of using only those that are more recurrent and thus presumably more stable; from OWL-S-side, the management of the knowledge base to ensure that the rules are SWRL DL-safe since, if two services consume the same last resource in the knowledge base at the same time, one of them will be invalid (information that may not be updated during the composition phase). Having investigated these aspects, a valid indication of the robustness of the proposed method can be obtained.

Finally, the need to manually write associations between Prolog predicates and OWL ontologies opens the way for ambitious conjectures in which the same operations are performed automatically or semi-automatically. This would require both the ability to handle and index large amounts of remote ontologies, but more importantly, to exploit complex natural language processing algorithms to detect, or at least suggest, potential correct associations. This perspective, although beyond the scope of this paper, would represent a remarkable breakthrough, which is able to intersect and merge, under a common goal, several areas of Computer Science research.

Author Contributions: Conceptualization, D.R. and S.F.; data curation, D.R. and S.F.; formal analysis, D.R. and S.F.; investigation, D.R. and S.F.; methodology, D.R. and S.F.; resources, D.R. and S.F.; software, D.R. and S.F.; supervision, D.R. and S.F.; validation, D.R. and S.F.; writing—original draft, D.R. and S.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the Apulia Region through the Research for Innovation—REFIN' initiative [grant number 48B6D67E].

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FOL	First-Order Logic.
ILP	Inductive Logic Programming.
BPM	Business process management.
WFM	Workflow Management.
PAIS	Process-aware information systems.
BPMN	Business Process Management Notation.
WSDL	Web Service Description Language.
URI	Uniform Resource Location.
XML	eXtensible Markup Language.
RDF	Resource Description Framework.
OWL	Web Ontology Language.
OWL-S	OWL for services.
SWRL	Semantic Web Rule Language.
WoMan	Workflow Manager.

References

1. Van der Aalst, W.M.P. *Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management*; Springer: Berlin/Heidelberg, Germany, 2004. [CrossRef]
2. Weske, M. *Business Process Management: Concepts, Languages, Architectures*; Springer: Berlin/Heidelberg, Germany, 2007. [CrossRef]
3. Van der Aalst, W.M. Business process management: A comprehensive survey. *Int. Sch. Res. Not.* **2013**, *2013*, 507984. [CrossRef]
4. Dumas, M.; van der Aalst, W.M.P.; ter Hofstede, A.H.M. (Eds.) *Process-Aware Information Systems: Bridging People and Software Through Process Technology*; Wiley: Hoboken, NJ, USA, 2005. [CrossRef]
5. van der Aalst, W.M.P. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed.; Springer Publishing Company: Berlin/Heidelberg, Germany, 2011.
6. Hanga, K.M.; Kovalchuk, Y.; Gaber, M.M. A Graph-Based Approach to Interpreting Recurrent Neural Networks in Process Mining. *IEEE Access* **2020**, *8*, 172923–172938. [CrossRef]
7. Yan, X.; She, D.; Xu, Y.; Jia, M. Deep regularized variational autoencoder for intelligent fault diagnosis of rotor–bearing system within entire life-cycle process. *Knowl.-Based Syst.* **2021**, *226*, 107142. [CrossRef]
8. Yan, X.; She, D.; Xu, Y. Deep order-wavelet convolutional variational autoencoder for fault identification of rolling bearing under fluctuating speed conditions. *Expert Syst. Appl.* **2023**, *216*, 119479. [CrossRef]
9. Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D.; Patel-Schneijder, P.; Stein, L.A. OWL Web Ontology Language Reference. Recommendation, World Wide Web Consortium (W3C). 2004. Available online: http://www.w3.org/TR/owl-ref/ (accessed on 10 November 2023).
10. OMG. *Business Process Model and Notation (BPMN), Version 2.0*; Technical Report; Object Management Group: Needham, MA, USA, 2011.
11. Francescomarino, C.D.; Ghidini, C.; Rospocher, M.; Serafini, L.; Tonella, P. Semantically-Aided Business Process Modeling. In Proceedings of the Semantic Web—ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, 25–29 October 2009; Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5823, pp. 114–129. [CrossRef]
12. Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; Patel-Schneider, P. (Eds.) *The Description Logic Handbook*; Cambridge University Press: Cambridge, UK, 2003.
13. Ferilli, S.; Redavid, D.; Angelastro, S. Activity Prediction in Process Management Using the WoMan Framework. In Proceedings of the Advances in Data Mining. Applications and Theoretical Aspects—17th Industrial Conference, ICDM 2017, New York, NY, USA, 12–13 July 2017; Perner, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10357, pp. 194–208. [CrossRef]

14. Hahmann, T.; II, R.W.P. Automatically Extracting OWL Versions of FOL Ontologies. In Proceedings of the Semantic Web-ISWC 2021—20th International Semantic Web Conference, ISWC 2021, Virtual Event, 24–28 October 2021; Proceedings; Hotho, A., Blomqvist, E., Dietze, S., Fokoue, A., Ding, Y., Barnaghi, P.M., Haller, A., Dragoni, M., Alani, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; Volume 12922, pp. 252–269. [CrossRef]
15. Flügel, S.; Glauer, M.; Neuhaus, F.; Hastings, J. When one Logic is Not Enough: Integrating First-order Annotations in OWL Ontologies. *arXiv* **2022**, arXiv:2210.03497. [CrossRef]
16. Lopes, C.; Knorr, M.; Leite, J. NoHR: Integrating XSB Prolog with the OWL 2 Profiles and Beyond. In Proceedings of the 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, 3–7 June 2019; Balduccini, M., Janhunen, T., Eds.; Springer: Cham, Switzerland, 2017; pp. 236–249.
17. Costa, N.; Knorr, M.; Leite, J. Next Step for NoHR: OWL 2 QL. In Proceedings of the The Semantic Web—ISWC 2015, the 14th International Semantic Web Conference, Bethlehem, PA, USA, 11–15 October 2015; Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., et al., Eds.; Springer: Cham, Switzerland, 2015; pp. 569–586.
18. Zese, R.; Cota, G. Optimizing a tableau reasoner and its implementation in Prolog. *J. Web Semant.* **2021**, *71*, 100677. [CrossRef]
19. Di Martino, B.; Graziano, M.; Colucci Cante, L.; Esposito, A.; Epifania, M. Application of Business Process Semantic Annotation Techniques to Perform Pattern Recognition Activities Applied to the Generalized Civic Access. In Proceedings of the 16th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2022), Online, 29 June–1 July 2022; Barolli, L., Ed.; Springer: Cham, Switzerland, 2022; pp. 404–413.
20. Di Martino, B.; Colucci Cante, L.; Esposito, A.; Graziano, M. A tool for the semantic annotation, validation and optimization of business process models. *Software Pract. Exp.* **2023**, *53*, 1174–1195. [CrossRef]
21. Rebmann, A.; van der Aa, H. Enabling semantics-aware process mining through the automatic annotation of event logs. *Inf. Syst.* **2022**, *110*, 102111. [CrossRef]
22. Ardito, C.; Caivano, D.; Colizzi, L.; Verardi, L. BPMN Extensions and Semantic Annotation in Public Administration Service Design. In *Human-Centered Software Engineering*; Bernhaupt, R., Ardito, C., Sauer, S., Eds.; Springer: Cham, Switzerland, 2020; pp. 118–129.
23. Samuel, K.; Obrst, L.; Stoutenburg, S.; Fox, K.; Franklin, P.; Johnson, A.; Laskey, K.J.; Nichols, D.; Lopez, S.; Peterson, J. Translating OWL and semantic web rules into prolog: Moving toward description logic programs. *Theory Pract. Log. Program.* **2008**, *8*, 301–322. [CrossRef]
24. Elenius, D. SWRL-IQ: A Prolog-based Query Tool for OWL and SWRL. In Proceedings of the OWL: Experiences and Directions Workshop 2012, Heraklion, Greece, 27–28 May 2012; Volume 849. Available online: http://ceur-ws.org/Vol-849 (accessed on 10 November 2023).
25. Musen, M.A. The Protégé Project: A Look Back and a Look Forward. *AI Matters* **2015**, *1*, 4–12. [CrossRef] [PubMed]
26. Sagonas, K.F.; Swift, T.; Warren, D.S. The XSB Programming System. In Proceedings of the Workshop on Programming with Logic Databases. In Conjunction with ILPS, Vancouver, BC, Canada, 30 October 1993; Ramakrishnan, R., Ed.; University of Wisconsin: Madison, WI, USA, 1993; Volume 1183, p. 164.
27. Laera, L.; Tamma, V.A.M.; Bench-Capon, T.J.M.; Semeraro, G. SweetProlog: A System to Integrate Ontologies and Rules. In Proceedings of the Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004, Hiroshima, Japan, 8 November 2004; Antoniou, G., Boley, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3323, pp. 188–193. [CrossRef]
28. van der Aalst, W.M.P. Process mining. *Commun. ACM* **2012**, *55*, 76–83. [CrossRef]
29. Ferilli, S. Woman: Logic-based workflow learning and management. *IEEE Trans. Syst. Man Cybern. Syst.* **2014**, *44*, 744–756. [CrossRef]
30. Ferilli, S.; Esposito, F.; Redavid, D.; Angelastro, S. Extended Process Models for Activity Prediction. In Proceedings of the 26th International Symposium, ISMIS 2022, Cosenza, Italy, 3–5 October 2022; Kryszkiewicz, M., Appice, A., Ślézak, D., Rybinski, H., Skowron, A., Raś, Z.W., Eds.; Springer: Cham, Switzerland, 2017; pp. 368–377.
31. Petri, C.A.; Reisig, W. Petri net. *Scholarpedia* **2008**, *3*, 6477. [CrossRef]
32. van der Aalst, W.M.P. The Application of Petri Nets to Workflow Management. *J. Circuits Syst. Comput.* **1998**, *8*, 21–66. [CrossRef]
33. van Dongen, B.F.; de Medeiros, A.K.A.; Wen, L. Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. *Trans. Petri Nets Other Model. Concurr.* **2009**, *2*, 225–242. [CrossRef]
34. van der Aalst, W.; Carmona, J.; Chatain, T.; van Dongen, B. A Tour in Process Mining: From Practice to Algorithmic Challenges. In *Transactions on Petri Nets and Other Models of Concurrency XIV*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 1–35. [CrossRef]
35. Maggi, F.M. Declarative Process Mining. In *Encyclopedia of Big Data Technologies*; Sakr, S., Zomaya, A.Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2019. [CrossRef]
36. Muggleton, S.H. Inductive Logic Programming. *New Gener. Comput.* **1991**, *8*, 295–318. [CrossRef]
37. Cropper, A.; Dumancic, S.; Evans, R.; Muggleton, S.H. Inductive Logic Programming at 30. *Mach. Learn.* **2022**, *111*, 147–172. [CrossRef]
38. Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. *Sci. Am.* **2001**, *284*, 34–43. [CrossRef]

39. Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. Web Services Description Language (WSDL) 1.1. W3C Note, World Wide Web Consortium, 15 March 2001. Available online: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (accessed on 10 November 2023).
40. Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; et al. OWL-S: Semantic Markup for Web Services. 2004. W3C Member Submission, 22 November 2004. Available online: http://www.w3.org/Submission/OWL-S/ (accessed on 10 November 2023).
41. Horrocks, I.; Patel-Schneider, P.F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available online: http://www.w3.org/submissions/SWRL/ (accessed on 10 November 2023).
42. Redavid, D.; Iannone, L.; Payne, T.R.; Semeraro, G. OWL-S Atomic Services Composition with SWRL Rules. In Proceedings of the Foundations of Intelligent Systems, 17th International Symposium, ISMIS 2008, Toronto, ON, Canada, 20–23 May 2008; An, A., Matwin, S., Ras, Z.W., Slezak, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 4994, pp. 605–611. [CrossRef]
43. Redavid, D.; Ferilli, S.; Esposito, F. SWRL Rules Plan Encoding with OWL-S Composite Services. In Proceedings of the Foundations of Intelligent Systems—19th International Symposium, ISMIS 2011, Warsaw, Poland, 28–30 June 2011; Kryszkiewicz, M., Rybinski, H., Skowron, A., Ras, Z.W., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6804, pp. 476–482. [CrossRef]
44. Ferilli, S.; De Carolis, B.; Pazienza, A.; Esposito, F.; Redavid, D. An Agent Architecture for Adaptive Supervision and Control of Smart Environments. In Proceedings of the 2015 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), Angers, France, 11–13 February 2015.
45. Horrocks, I.; Parsia, B.; Patel-Schneider, P.F.; Hendler, J.A. Semantic Web Architecture: Stack or Two Towers? In Proceedings of the Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, 11–16 September 2005; Fages, F., Soliman, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3703, pp. 37–41. [CrossRef]
46. Euzenat, J.; Shvaiko, P. *Ontology Matching*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 1–511. [CrossRef]
47. Lloyd, J.W. *Foundations of Logic Programming*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 1987.
48. Motik, B.; Sattler, U.; Studer, R. Query Answering for OWL-DL with rules. *J. Web Semant. Sci. Serv. Agents World Wide Web* **2005**, *3*, 41–60. [CrossRef]
49. Hayes, P.; Welty, C. Defining N-ary Relations on the Semantic Web. Working Group Note, World Wide Web Consortium (W3C). 2006. Available online: http://www.w3.org/TR/swbp-n-aryRelations (accessed on 10 November 2023).
50. Da Silva, A.F.; Costa, V.S. The Design and Implementation of the YAP Compiler: An Optimizing Compiler for Logic Programming Languages. In Proceedings of the Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, 17–20 August 2006; Etalle, S., Truszczynski, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4079, pp. 461–462. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.