Automated Data Management Workflow Generation with Ontologies and Planning

PuK 2016 submission

University of Freiburg, Germany
{bwright,mattmuel}@informatik.uni-freiburg.de

Abstract. When working with data management systems, it is often required to specify and model the data within the system. Ontologies are a widely used method of defining data and its relations within a system by defining concepts, properties, and roles. In addition to the definition of the data structures, workflows are required for adding new knowledge to the ontologies. We will show how AI planning can be used to derive such workflows from the ontology, enforcing the closed world assumption required for data management systems.

Keywords: planning, data management, workflow management, ontologies

1 Introduction

Data management deals with all issues arising during the life cycle of data as a resource. One major issue is the definition of the actual data: Which data types are there and what kind of information is to be stored about them. During the development of a data management system for archiving research data, we were confronted with two of these aspects: How should the data be modelled, and how should the workflows for archiving this data be managed. As a solution to the first issue, ontologies where chosen. Ontologies can be used to model the structure of data, the relationships between pieces of data, and its context within an organization (e. g., who created the data). An ontology consists of two parts, the *TBox* which holds the concept, role, and property definitions, and the *ABox* which holds the actual instances of the concepts, roles, and properties. In the *TBox*, concepts and properties are used to model data types. For instance, the concept *Book* can be defined and the property *title* can be added to this concept. Then roles can be added to put the concepts into relations with each other, e. g., a role *isAuthorOf* which puts a person and a book into an authorship relation. In addition to the definition of the data types, *workflows* are needed to add the data to a data management system, and to perform administrative tasks on them. A workflow in the context of ontology-driven data management can be defined as sequence of *assertRole*, *create*, *select* statements which transform the initial *ABox* A into a target *ABox* A' such that $A' \equiv A \cup B$ with B a set of target assertions. Although ontologies operate in an open world context, we require each object that is used for an assertion to be created prior to its first use. This enforces a closed world paradigm required for data management. This requirement introduces acyclic dependencies between the assertions in the workflow, generating a natural ordering of actions to be taken. Additionally, some actions will be taken by humans, introducing some nondeterministic effects to the workflow, which necessitates branching depending on the user input. This has the effect that the sequence in which assertions have to occur is non-trivial. Hoffmann et al. [12] showed that deriving workflows can be achieved by means of AI planning. However, they used SAP's proprietary SAM (Status and Action Management) model for generating the planning domain. We will show how knowledge from an ontology can be translated into a planning task with the goal of finding a sequence of actions which creates the aforementioned assertions. Additionally to the non-trivial sequence of actions, data management systems can consist of hundreds of data types, making the task of modelling each workflow manually very labour intensive. Therefore, an automated approach was chosen for our archival system mentioned above.

Technically speaking, we define a *state* of a workflow as a tuple $s = \langle T, A, C \rangle$, where T is a *TBox*, A is an *ABox*, and C is a set of individuals that were already *created*. We use C to allow working in a closed-world setting where objects need to be created before any assertion about them can be made. Then, a *problem instance* P is an initial state $s_0 = \langle T, A_0, C_0 \rangle$ of a workflow plus a *target assertion* a_0, which is a triple $(i, j) : r$ consisting of two individual names i and j and a role name r. For simplicity, we assume that the initial *TBox* T is fixed and cannot be modified, and that the initial *ABox* A_0 is empty. C_0 may or may not contain any elements. We allow the following modifications of a state $s = \langle T, A, C \rangle$ as part of the workflow we want to construct: A *create* statement creates a new individual not yet in C and adds it to C, and an *assertRole* statement adds a new role assertion to A putting two individuals previously created into a role relationship (for a specific role name r). *Create* statements are only allowed if the individual to be created is not yet in C, and *assertRole* statements are only allowed if the relevant individuals have been created before. A *solution* to P is then a sequence $a_0, \dots, a_{n-1}$ of *assertRole* and *create* statements producing states $s_1, \dots, s_n$ such that (i) statement a_i is allowed in state s_i, that (ii) state s_{i+1} results from applying statement a_i to state s_i, $i = 0, \dots, n-1$, and that (iii) in state $s_n = \langle T, A_n, C_n \rangle$, the target assertion holds, i. e., that $A_n \models a_0$.

Notice that this only allows *linear* workflows where all relevant individuals are either already created (and known to be created) prior to the workflow execution, or explicitly created during the workflow. A more expressive framework would additionally allow *select* statements as part of a workflow, modelling that a user attempts to select a desired individual from a list of known individuals. However, this user interaction would introduce nondeterminism and hence possibly *branching* workflows that we do not yet support, but rather consider to be future work.

In order to obtain a solution to P, we translate P to a planning task $\Pi(P)$ that is solvable if and only if P has a solution and such that every plan $\pi(\Pi(P))$

for $\Pi(P)$ translates back to a solution $\tau(\pi(\Pi(P)))$ to P using an appropriate plan-to-statement-sequence translation function τ.

Motivating Scenario. Ontologies can be used to model a wide range of knowledge. This leads to the issue that for each new ontology, applications working with this knowledge need to be adapted. These applications usually provide some means of adding or manipulating the knowledge in the ontology, implementing workflows defined by individuals or organisations. Manually creating these workflows for large knowledge bases can be very tedious. Therefore, means of generating these workflows automatically provides a major improvement to such systems. Our main motivation arose during the development of a software system for long-term preservation of research data. During this, the requirement for modelling document types together with workflows for their creation arose. Consider for instance an ontology modelling the document type *Article*, with roles putting the article into context, such as a role *isAuthorOf* relating persons to articles, or a role *isFundingAgencyOf* relating funding agencies to articles. One workflow could now describe the action required for adding a new article to the knowledge base. This could consist of uploading a file and selecting a set of authors from a list of people already in the system. Similarly, a list of funding agencies can be selected. Additionally, new people or agencies might need to be added to the knowledge base. What we want to achieve is an automated way of deriving such workflows from the knowledge stored in a given ontology. We would like to emphasize at this point, that this paper presents our preliminary results, and should serve as starting point for discussions and feedback.

2 Related Work

As mentioned in the introduction, Hoffmann et al. [12] showed that AI planning can be used to create workflows for business process management. However, they used SAP's internal SAM model for modelling the business objects, where each business object is associated with status variables and possible actions. In contrast, we do not a-priori associate elements from our ontology to possible actions in our planning domain, but rather generate the actions during the translation process from ontology to PDDL. Bouillet et al. [4] presented a framework which allows the problem to be modelled using OWL ontologies. However, each action is modelled explicitly as an ontology pattern. Even though this provides a very flexible way of defining problem domains, we focus on generating sequences of assertions without their explicit modelling. To our knowledge, no previous work exists focusing on deriving plans for adding new knowledge to an existing ontology, based solely on the existing ontology. A different approach to adding new knowledge to an existing ontology is show in Calvanese et al. [5], who show how actions defined in Knowledge and Action Bases, more precisely state bounded KABs, are used in ADL planning for manipulating the original ontology. However they use a defined set of actions which are on the one hand more flexible, but also introduce additional complexity during the knowledge base modelling. Related work also contains work on automated web service composition via planning [3], discussing which in detail would go beyond the scope of this paper.

3 Background

In this section, we introduce the required background to our work, starting with the definitions used in ontologies followed by an introduction to planning.

3.1 Ontologies

Ontologies can be used to structure knowledge by defining concepts and relationships between concepts, adding structure to otherwise unstructured data. An ontology thereby consists of two parts: The *TBox* holding the definitions outlining the terminology of the knowledge base, and the *ABox* consisting of the elements representing the actual state of knowledge. The terms in the *TBox* are *concepts* which identify individuals as instances of given concepts, and *roles* denoting binary relationships between concepts. In the *ABox*, the *concepts* and *roles* are instantiated in the form of individuals (*concept assertions*) and roles (*role assertions*) [1]. During ontology development, the terms *object property* and *data property* will occur. *Object properties* refer to roles regarding two individuals of the type *concept assertions*, whereas *data properties* refer to relationships between *concept assertions* and *value assertion* (String, Integer, Date, ...). In the remainder of this paper, the term property is only used in the context of the latter. And individuals can be of the form *concept assertion* or a *value assertion.*

Description Language. The description language used in this paper will be $\mathcal{FL}_f^-$, which supports the following concepts: A (atomic concepts), $\top, \bot$ (top and bottom concept), $C \sqcap D$ (intersection), $\forall R.C$ (value restriction), $\exists R.\top$ (limited existential restriction). Additionally, functional roles are supported. Expanding to a more expressive description logic will be part of future work.

3.2 Planning

We translate the search for a successful sequence of assertions to the search for a plan in a corresponding planning task. To model planning tasks, we use the following framework: A planning task consists of a finite set of finite-domain state variables, a finite set of operators with preconditions and effects, an initial state description and a goal condition. To simplify the translation, we translate to a rather expressive fragment of ADL [18] and refer the reader to the literature on how to compile this fragment further down to STRIPS [8] or SAS$^+$ [2]. We allow the following ADL features: negation, disjunction, universal and existential quantification in preconditions, universal and conditional effects, equality, and typing. Our notation in the examples below should be largely self-explanatory. Conditional effects are written as $c \rhd e$ where c is the effect condition and e is the effect.

The semantics of planning tasks is as usual, i. e., a plan is a sequence of applicable operators transforming the initial state to a state satisfying the goal condition. Details can be found in the literature [9, 16]. Notice that we explicitly do not make an open-world assumption on the planning side, but rather assume a closed world as usual in classical planning. This guarantess that planners such as Fast Downward [10] support the language we use. In the following, we will freely switch between schematic and grounded representations, and, for notational convenience, use schematic representations to specify operators resulting from our compilation.

3.3 Open World or Closed World

The closed-world assumption states that everything which is true is also known to be true. In contrast, the open-world assumption states that if something is not known to be true, it may or may not be true. No precise statement can be made. When dealing with ontologies, the open-world assumption is made, which means that knowledge that is not stated in the *ABox* may simply be missing. However, when we deal with AI planning, only knowledge that is known to be true can be treated as being true, i. e., planning makes a closed-world assumption. This has to be taken in to account when translating from ontologies to planning. How this is achieved will be discussed in Section 5.1.

4 Modelling Ontologies for Data Management

To define the data types, first a concept representing this data type needs to be specified. Then all properties and relations need to be defined. This can be achieved using $\mathcal{FL}_f^-$ which allows the definition of properties, roles, and concepts. Some of the properties can be defined as being functional, restricting the amount of these properties each individual of a concept can be associated with to one. Roles are defined in the form of triples consisting of the *domain*, *role*, and *range*, which can be interpreted as concept *domain* is in relation *role* with concept *range*. Instances of concepts and properties shall be called *individuals* from here after. Even though $\mathcal{FL}_f^-$ seems to be a very limited DL, it is expressive enough to define the data types required in our data management system. Additionally to the constructors mentioned in Section 3.1, we can use $\sqsubseteq$ and $\equiv$(concept inclusion and concept equivalence) to create our data type definitions. This allows concepts to be created, properties assigned to these concepts, and concepts to be put into relation with each other. Consider an example: Defining the concepts *Person* and *Book*, properties *name* and *title*, and the role *isAuthor*, we can define a simple ontology describing the data type *Book* as shown in Fig. 1. Yellow boxes represent concepts, and green boxes represent property value types that are needed to define data properties. Solid connections represent role relations (individuals of type *Person* can be in authorship relations to individuals of type *Book*), and similarly, dashed connections represent data properties of the connected concepts (individuals of type *Person* have a name, which is a character string, and similarly, individuals of type *Book* have a title). The numbers 1 and n used as edge labels denote whether the pertinent roles/properties are required to be functional (1) or not (n). Intuitively, there are many ways of defining the same semantic concept or relation. Therefore it is advised to use a generally accepted standard for modelling an ontology. One such standard applicable to data management is the Dublin Core standard [7] which can serve as a starting point, refining it where necessary. Lenzerini [13] gives a nice introduction to the use of ontologies in data-management systems, together with some of the arising issues.

5 From Ontology to Planning Task

We want to faithfully translate the knowledge encoded in a given ontology to a planning task. In this section, the very simple ontology shown in Fig. 1 is used to illustrate the translation from ontology to planning task.

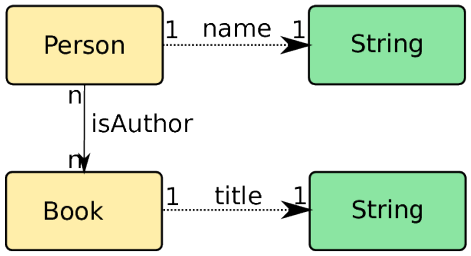

Fig. 1. Sample Ontology

As we are only interested in generalized workflows, we consider the ABox to be empty in the initial state. This will guarantee that we create workflows independently of the current knowledge in the ABox. Using assertions from the ABox would result in workflows specially targeting certain instances from the ABox, making the resulting workflow instance specific. Additionally, editing or deleting assertions from the ontology is currently not supported, and is subject to further research.

5.1 Creating the Planning Domain

Starting from the target assertion, all relevant concepts, properties and roles are identified. In this context, relevant means that the concept, role, or property is in a role or property relation to the target assertion or an element previously identified as relevant. Relevancy is calculated recursively until reaching the top element by traversing all super-class relations encountered.

Concepts and Individuals. For each identified concept C, a type t_C is added to the planning task, and an action a_C for the creation of individuals of this type is added to the set of operators, which marks an individual c of type t_C as created by setting a corresponding predicate $\mathit{created}(c)$ to true. This action is needed to bridge the gap between open-world semantics (ontologies) and closed-world semantics (planning). Additionally, the type t_C is populated with a predefined number of objects by mentioning them in the initial state. Those objects may or may not already be *created*, depending on the current state of the ontology. As long as they are not *created*, nothing can be asserted about them. Additionally, for each role and property, a constant of type *role* is added to the set of domain constant symbols. It is noteworthy at this point that we use planning types to both encode different concepts as well as the “meta types” role, concept, property and individual. We do this by creating a typing system defining assertion, role, concept, and individual as elements of type object, and create derived types for each element from the ontology.

When dealing with complex concepts such as $C \sqsubseteq A \sqcap B$ we create types for C, A, and B. Creating an individual of type A and of type B will result in the creation of two separate elements a and b. For the desired result here, it is required to create an individual of type C

Roles. In the ontology, roles are used to describe the relation in which two individuals stand. Similarly, data properties relate individuals to value types. For a uniform treatment, in the planning task, we deal with roles and data properties in the same way and hence focus on describing roles here. The only difference is that data properties are asserted during their concept's creation actions, and roles can be asserted individually. So, in the planning task, we introduce a type *role*, and for each role name r from the ontology, a constant object m_r of type *role*.

Assertions. Assertions define which information is explicitly present in the ontology. TBox assertions state what concepts and roles are defined, and ABox assertions define the concrete individuals and relations of the given concepts and roles stored in the ontology. For the planner to be able to deal with the notion of assertions, we introduce a new type *assertion*. For roles we then define a subtype *roleAssertion* which represents the ontology triples $(i, j) : r$ of individuals i and j and the role r, which can be interpreted as the individual i is r-related to j. For each possible role assertion from the subset of relevant roles we create an object a of type *roleAssertion* and relate the individuals i and j as well as the role r to it. This is achieved by adding the following statements to the initial state of the planning problem: $\mathit{domain}(a, i)$, $\mathit{range}(a, j)$, and $\mathit{role}(a, r)$, which corresponds to the domain, range and role defined in the ontology. Notice that we do not need a type *conceptAssertion*, since we do not allow new concept assertions to be made during the planning process. Rather, each individual is fixed to belong to a certain set of concepts in the initial state of the planning task and this can never change. This process introduces many *roleAssertions* which might never be used, however, since we limit the possible assertions to the ones identified as relevant in the previous step, the amount of assertions to be taken in to account during planning is bounded by the number of relevant assertions, not to the size of the whole ontology. This greatly reduces the overall complexity of the planning task.

Actions. Actions will differ for each ontology, as they are dependent on the concepts, properties and roles. However one action will be consistent over all ontologies and is required for the above mentioned role assertions. An action *assertRole* is therefore defined. It takes as parameters two individuals i and j, and one role name r. It then selects an existing *roleAssertion* which matches these parameters, and marks it as *asserted*, indicating that, as a result of this action, the role is asserted to the ontology's *ABox*.

$$\begin{aligned}
\texttt{Operator} &= \textit{assertRole}(i : \textit{individual}, r : \textit{role}, j : \textit{individual}) \\
\texttt{pre} &= \textit{created}(i) \wedge \textit{created}(j) \\
\texttt{eff} &= \forall a : \textit{roleAssertion} \\
&\qquad \big((\textit{domain}(a, i) \wedge \textit{role}(a, r) \wedge \textit{range}(a, j)) \\
&\qquad\quad \rhd \textit{asserted}(a)\big)
\end{aligned}$$

For each subtype of individual, a create action is required. The create actions for properties are fairly straightforward as the action takes an individual of the creation type as a parameter and marks it as *created* in its effect. This ensures that all properties used in later create or assert actions are created prior to their use, enforcing the required closed-world paradigm.

$$\begin{aligned}
\texttt{Operator} &= \textit{createProperty}(p : \textit{property}) \\
\texttt{pre} &= \neg\textit{created}(p) \\
\texttt{eff} &= \textit{created}(p)
\end{aligned}$$

Each concept from the ontology requires its own create action, which also takes all its functional properties as parameters. This ensures that the correct properties are assigned to the new instance of the concept. The precondition of the action checks if the passed properties are created prior to their use and are not in use by a different *roleAssertion*, thus not assigned to another individual. The effect of this action is that each property individual is asserted as a *roleAssertion* associating the property to the concept individual, and the new concept individual is marked as *created*. At this point it should be stated that when dealing with concept intersection, it is required to create the exact type. Let $A \sqsubseteq B \sqcap D$, creating an individual of type B and D is interpreted as two separate individuals, and thus not an instance of A. To create an instance of type A, it has to be created explicitly. On the other hand, creating an element of type A will have all properties associated with B and D.

$$
\begin{aligned}
&\texttt{Operator} = \mathit{createConcept}(c : t_C, [p_1 : P_1, \ldots, p_n : P_n])\\
&\quad \texttt{pre} = \neg \mathit{created}(c) \wedge \mathit{created}(p_1) \wedge \cdots \wedge \mathit{created}(p_n) \wedge\\
&\qquad \neg \exists i_1 : \mathit{individual}, a_1 : \mathit{roleAssertion}\\
&\qquad\quad \big(\mathit{domain}(a_1, i_1) \wedge \mathit{role}(a_1, \mathit{role}_{p_1}) \wedge \mathit{range}(a_1, p_1) \wedge \mathit{asserted}(a_1)\big) \wedge\\
&\qquad \exists i_1 : \mathit{individual}, a_1 : \mathit{roleAssertion}\\
&\qquad\quad \big(\mathit{domain}(a_1, i_1) \wedge \mathit{role}(a_1, \mathit{role}_{p_1}) \wedge \mathit{range}(a_1, p_1)\big) \wedge\\
&\qquad \ldots\\
&\qquad \neg \exists i_n : \mathit{individual}, a_n : \mathit{roleAssertion}\\
&\qquad\quad \big(\mathit{domain}(a_n, i_n) \wedge \mathit{role}(a_n, \mathit{role}_{p_n}) \wedge \mathit{range}(a_n, p_n) \wedge \mathit{asserted}(a_n)\big) \wedge\\
&\qquad \exists i_n : \mathit{individual}, a_n : \mathit{roleAssertion}\\
&\qquad\quad \big(\mathit{domain}(a_n, i_n) \wedge \mathit{role}(a_n, \mathit{role}_{p_n}) \wedge \mathit{range}(a_n, p_n)\big)\\
&\quad \texttt{eff} = \mathit{created}(c) \wedge\\
&\qquad \forall a_1 : \mathit{roleAssertion}\\
&\qquad\quad \big((\mathit{domain}(a_1, c) \wedge \mathit{role}(a_1, \mathit{role}_{p_1}) \wedge \mathit{range}(a_1, p_1)) \rhd \mathit{asserted}(a_1)\big) \wedge\\
&\qquad \ldots\\
&\qquad \forall a_n : \mathit{roleAssertion}\\
&\qquad\quad \big((\mathit{domain}(a_n, c) \wedge \mathit{role}(a_n, \mathit{role}_{p_n}) \wedge \mathit{range}(a_n, p_n)) \rhd \mathit{asserted}(a_n)\big)
\end{aligned}
$$

Here, role_{p_n} is the role associated with the property p_n, t_C the type of the concept C, and $P_1, \ldots P_n$ are the types of the properties. In the *precondition* section it is first ensured that there are no assertions with the passed parameter and any other individual or role, which is already marked as asserted. This ensures that the concrete property has not yet been used. Secondly the existence of such a *roleAssertion* is checked. This guarantees that the effect of the action can later mark the *roleAssertion* as asserted.

5.2 Creating the Planning Task

This section will discuss how the planning problem can be generated from the target assertion and the ontology. Starting from the target assertion, all related concepts, roles, and properties are identified. For each of these identified elements, a corresponding object (representing an individual of that type) is added to the set of task-dependent constants and initialized in the initial state s_0. We can see now that all concepts are treated as domain-dependent constants, and individuals are treated as task-dependent constants. Planning objects for all elements required for the target assertion together with all its related elements (properties, roles, concepts) are then initialized in s_0 by associating the properties to their respective concept instance. This is achieved by adding the corresponding *roleAssertions* to s_0.

Similarly the roles are constructed by adding objects for domain, range, and role assertion to the planning task, and initialising them in s_0, whereby existing range or domain objects must be taken in to account. This is achieved by creating unique identifiers for each object. By doing this for all concepts, roles, and properties encountered during the identification of related elements of the target assertion, we ensure that in the goal state $s_\star$ all objects will have been created or asserted, ensuring that the newly asserted knowledge holds even under the closed-world assumption.

If the target assertion was defined as being a concept assertion, then we add this assertion to the goal description of the problem as $\mathit{created}(a_0)$, equally, if the target assertion was a role assertion, then we add it to the goal state as $\mathit{asserted}(a_0)$.

An Example Plan. Consider again the very simple ontology from Figure 1 and the goal of annotating a person as an author of a book. The target assertion would then be $(\mathit{person1}, \mathit{book1}) : \mathit{isAuthor}$ for arbitrary but fixed individuals *person1* of type *Person* and *book1* of type *Book*. Executing our tool as described above would identify *Person*, *name*, *Book*, and *title* as relevant elements, and require $(\mathit{person1}, \mathit{book1}) : \mathit{isAuthor}$ to be asserted in the goal state. Four individuals would be generated in the initial state of the problem, which the planner would try to set as *created*. These individuals would be a *Person*, a *Book*, a *title*, and a *name*. Additionally, three role assertions would be generated mapping the *name* and *title* to the respective elements, and one for the target assertion $(\mathit{person1}, \mathit{book1}) : \mathit{isAuthor}$. A resulting plan would be as follows:

```
createname unknownname
create person1 unknownname
createtitle unknowntitle
createbook book1 unknowntitle
assertRole person1 isauthor book1
```

This plan can be interpreted as:

1. Create a Person *person1* with the name `unknownname`.
2. Create a Book *book1* with the title `unknowntitle`.
3. Define *person1* as author of *book1*.

6 From Plan to Workflow

A plan for the planning task resulting from our translation is a sequence of *create* and *assert* statements. These can be used to define the steps in a workflow. Each *create* step either creates a concept instance or a property. The *create* actions of properties can be interpreted as user input. Therefore the concept instance creation steps can be grouped together with their respective property creations to create a single workflow step. Due to the fact that all objects (concept instances and properties) are created prior to their usage, *assert* actions can be executed without further user input, and can be treated as separate workflow step. These *assert* steps will usually then be executed automatically, but depending on the application might be triggered manually.

Workflow Definition. Many ways exist for formal definition of workflows, among them the XML Process Definition Language(XPDL) [19], and the Business Process Model and Notation (BPMN) [17]. Since the requirements of the data management system in development were very limited, a very lightweight workflow definition was developed. This simple system consists of steps to be executed in a linear workflow. For the workflow to be executed on multiple applications, a JSON encoding was chosen. Each step is hereby defined by a list of actions to be taken and a list of resulting ontology assertions. The sequence in which these steps need to be taken is encoded by a simple enumeration. For the addition of branching in the workflow, a list of follow-up steps may be added to each step, hereby replacing the simple numbering by unique identifiers.

7 Adding Nondeterministic Actions

In this section we will shortly discuss how the above ideas can be extended to provide the required nondeterminism, which is introduced by user interaction. In the simplest form this can be interpreted as the user being presented with two options for creating the same object. Let us say the user is supposed to state the author of a book. This can be achieved by selecting a person from a list of existing persons, or, if the required person is not on the list, by adding a new person to the ontology. In this case we have to model our *create* actions as nondeterministic actions, which instead of a single effect, have a set of possible effects. For each possible outcome of the action, a different plan has to be generated. This has the consequence that instead of an ordered list of assertions, we are presented with a tree of actions, with each action having multiple successors.

Such a tree or DAG structure that solves a planning task no matter which action outcomes take place is called a *strong plan* [6]. Notice that we really require *strong* plans, which are only allowed to branch, as opposed to *strong cyclic* plans, which are also allowed to have loops, in our scenario. Finding strong plans essentially boils down to AND/OR graph search, for which there are various approaches, including symbolic backward search as used by the MBP planner [6], informed forward search as used by CONTINGENT-FF [11] and MYND [14], and offline replanning with generalization of subplans as successfully proposed as part of the PRP planner [15]. Here, we will not go into the details of these algorithms and planners, but rather point out that we can choose any such approach and use it as a black box to obtain a solution. Integration into our framework is considered future work.

8 Preliminary Results

In this paper, we showed that AI planning is suitable for generating workflows for managing knowledge stored and modelled in an ontology. During the development of this paper, multiple ontologies where designed for testing, including the ontology used in our archival system. During our experiments we were able to fully automatically generate the workflow for asserting a new individual comprising of 20 data properties, 12 roles and 13 related concept instances into our test ontology. The resulting workflows generated showed great promise for real-world application. Given an ontology O, we were able to generate an ordered list of assertions achieving the target assertion, by translating the ontology and the target assertion into a planning task Π, finding a plan π for Π, and converting this plan back to a workflow achieving the target assertion.

9 Future Work

As mentioned earlier, we would like to extend this work to a more expressive description logic supporting full $\mathcal{ALF}$ functionality. The main issue here will be identifying the preconditions and effects of the *create* and *assert* actions. Additionally to adding more functionality on the ontology side, migrating to a more widely used and expressive workflow modelling language such as XPDL or BPMN would increase the usability in more general scenarios.

In real-world applications, workflows are often executed by multiple agents, with each agent having certain rights to execute parts of the workflow. Adding this to our system could be achieved by implementing this in a multi-agent planning scenario, with each agent having only the actions available, which the corresponding real-world agent would have the rights to execute.

Large-scale applications will usually have workflows that can be split into multiple smaller ones, which might be executed multiple times (by multiple agents). For this, sub-workflow detection needs to be executed, identifying workflows-sections common to multiple different workflows, and extracting these, so they can be reused multiple times.

10 Acknowledgements

This work has partly been supported by the German Research Foundation under grant EXC 1086.

References

1. Baader, F., Nutt, W.: Basic description logics. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.) The Description Logic Handbook, chap. 2, pp. 43–96. Cambridge University Press (2003)
2. Bäckström, C., Nebel, B.: Complexity results for SAS+ planning. Computational Intelligence 11, 625–656 (1995)

3. Bertoli, P., Pistore, M., Traverso, P.: Automated composition of web services via planning in asynchronous domains. Artificial Intelligence 174(3–4), 316–361 (2010)
4. Bouillet, E., Feblowitz, M., Liu, Z., Ranganathan, A., Riabov, A.: A knowledge engineering and planning framework based on owl ontologies. Proceedings of the Second International Competition on Knowledge Engineering (2007)
5. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: Plan synthesis for knowledge and action bases. In: Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI 2016). AAAI Press (2016), to appear
6. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence 147(1–2), 35–84 (2003)
7. Dublin Core Metadata Initiative: Dublin Core (April 2016), `http://http://dublincore.org/`
8. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(3), 189–208 (1971)
9. Geffner, H., Bonet, B.: A Concise Introduction to Models and Methods for Automated Planning (2013)
10. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research (JAIR) 26, 191–246 (2006)
11. Hoffmann, J., Brafman, R.: Contingent planning via heuristic forward search with implicit belief states. In: Proc. 15th International Conference on Automated Planning and Scheduling (ICAPS 2005). pp. 71–80 (2005)
12. Hoffmann, J., Weber, I., Kraft, F.M.: SAP Speaks PDDL: Exploiting a Software-Engineering Model for Planning in Business Process Management. Journal of Artificial Intelligence Research 44, 587–632 (2012)
13. Lenzerini, M.: Ontology-based data management. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management. pp. 5–6. CIKM '11, ACM, New York, NY, USA (2011), `http://doi.acm.org/10.1145/2063576.2063582`
14. Mattmüller, R., Ortlieb, M., Helmert, M., Bercher, P.: Pattern database heuristics for fully observable nondeterministic planning. In: Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS 2010). pp. 105–112 (2010)
15. Muise, C., McIlraith, S.A., Beck, J.C.: Improved non-deterministic planning by exploiting state relevance. In: Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012) (2012)
16. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice (2004)
17. Object Management Group: Business Process Model And Notation$^{\text{TM}}$(BPMN$^{\text{TM}}$) (April 2016), `http://www.omg.org/spec/BPMN/`
18. Pednault, E.: ADL: Exploring the middle ground between STRIPS and the situation calculus. In: Proc. 1st International Conference on Principles of Knowledge Representation and Reasoning (KR 1989). pp. 324–332 (1989)
19. Workflow Management Coalition: XPLD (April 2016), `http://www.xpdl.org/`