Towards workflow planning based on semantic eligibility

Timo Homburg, Pol Schumacher and Mirjam Minor

Goethe University, Frankfurt, Hessen 60325, Germany,
timo.homburg@stud.uni-frankfurt.de,
schumacher@cs.uni-frankfurt.de, minor@cs.uni-frankfurt.de

Abstract. A major problem in the research for new artificial intelligence methods for workflows is the evaluation. There is a lack of large evaluation corpora. Existing methods manually model workflows or use workflow extraction to automatically extract workflows from text. Both existing approaches have limitations. The manual modeling of workflows requires a lot of human effort and it would be expensive to create a large test corpus. Workflow extraction is limited by the number of existing textual process descriptions and it is not guaranteed that the workflows are semantically correct. In this paper we suggest to set up a planning domain and apply a planner to create a large number of valid plans. Workflows can be derived from plans. The planner uses a semantic eligibility function to determine whether an operator can be applied to a resource or not. We present a first concept and a prototype implementation in the cooking workflow domain.

1 Introduction

Workflows can be used to execute business processes. A workflow consists of a control-flow and a data-flow. A set of *activities* combined with *control-flow-structures* like sequences, parallel or alternative branches, and loops form the control-flow. A *non-sequential* control-flow denotes a control-flow which contains branches or loops. In addition, activities consume *resources* and create certain *products* which both can be physical matter (such as a component of a vehicle) or information. The data-flow describes the interaction of activities with resources and products. Different approaches aiming at different aspects of workflow had been presented, for example retrieval [1], clustering [2,3] or automatic adaptation [4]. A test repository of workflows is necessary for evaluation purposes. The creation of a repository with workflows that can be used for evaluation purposes is a challenging problem. A common approach is to handcraft workflows for a test repository [1,4]. Although these repositories contain high quality workflows a drawback is that it requires a lot of effort. Therefore it is not applicable for experiments which require a large number of workflows. Another approach for test set creation is the use of workflow extraction [5]. Workflow extraction is the transformation of textual process descriptions into formal workflow models [6]. The number of workflows which can be created by this approach is limited by the number of available textual process descriptions and the extracted workflows might contain errors. It is application dependent if automatically extracted workflows can be used as a test repository. We identified a research gap for a cost efficient method which can create a large number of semantically correct workflows. The problem of semantically correct workflows is

to restrict the set of workflows to those that are plausible with respect to user experience. This is achieved by re-using activities only with resources that occurred with this activity before. We introduce an eligibility function for operators that test whether an operator (activity) is applicable to a resource according to our plausibility expectations. An ontology is used to implement a semantic eligibility function.

Our idea is to use a planning approach to generate workflows. It receives a set of products as input and then generates the workflows which are applicable on this products set. To generate the workflows we use a forward chaining planner. A knowledge based eligibility function is used to determine if an operator is applicable in a specific state. The eligibility function has a similar role as the preconditions in classical STRIPS planning. The eligibility function queries an ontology if an operator can be applied to a product set. The ontology is constructed from existing workflows. The human effort for the construction of the planning domain can be reduced, because the ontology is filled with the empirical data from existing workflows.

In this paper we present a first concept of workflow planning. The paper is organized as follows. The next section introduces the workflow planning problem and the related concepts. The third section presents our planner. This section is followed by an example in the cooking domain. Related work on workflow planning is presented in the fifth section. The paper ends with a conclusion and an outlook on our future work.

2 Modelling

In this section we present our planning problem as Problem Description 1, describe its elements and build up connections of the planning problem to the concept of workflows. The initial state of the problem consists of a set of *r* available resources and a set of eligible operators which can be applied on a specific subset of *r*. The goal state is defined as the state in which no eligible method can be found for the given set of recources anymore. The operators are divided in four classes, as described in the subsection Operators.

2.1 State and Result

The planner, according to the STRIPS specification contains an initial (and potentially further modified) state(s) of resources and resource attributes. A resource attribute is defined as the modification of the name of a resource indicating a status change. To illustrate this we provide an example in Listing 1:

$$Have(water) \xrightarrow{heat} \neg Have(water) \wedge Have(water)[heated]$$

Listing 1. Modification of resource names

Here, the resource water is modified by the operator *heat* which adds the attribute *[heated]* to the resource water.
A state may contain which operators the resources including their attributes. Attributes essentially represent facts containing already applied operators, the availability of a recource and possible other facts worth noting during the planning process.

Problem Description 1: Planning Problem

Initial state: $\{Have(r) : r\ available\ resource\}$
$\{Eligible(r, o) : for\ all\ operator\ o\ applicable\ to\ resource\ r\}$
Goal state : $\forall r, o : Have(r) : (\neg Eligible(r, o) \wedge Have(r))$

1 //ActionTypes: 4 different operator types as explained below
2 $R_{curr} = \{r : r\ resource\ on\ which\ operator\ is\ applied\}$
3 **AttributeOperator** $(o(R_{curr})$,
4 PRECOND: $\forall r \in R_{curr} : Have(r) \wedge Eligible(r, o)$
5 EFFECT: $\forall r \in R_{curr} : \neg Have(r) \wedge \forall r \in R_{curr} : Have(r[o]))$
6 **MergingOperator** $(o(R_{curr})$,
7 PRECOND: $\forall r \in R_{curr} : Have(r) \wedge Eligible(r, o)$
8 EFFECT: $\forall r \in R_{curr} : \neg Have(r) \wedge Have(r_{new}))$
9 **AddingOperator** $((o(R_{curr})$,
10 PRECOND: $\forall r \in R_{curr} : Have(r) \wedge Eligible(r, o)$
11 EFFECT: $\neg Have(r_1) \ldots \neg Have(r_n) \wedge Have(r_n[r_1 \ldots r_{n-1}]))$
12 **RemovingOperator** $((o(R_{curr})$,
13 PRECOND: $\forall r \in R_{curr} : Have(r) \wedge Eligible(r, o)$
14 EFFECT: $\neg Have(r_n[r_1 \ldots r_{n-1}]) \wedge Have(r_1) \ldots Have(r_n))$

The result set is represented by a list of operators describing the way to accomplish a certain goal of a plan. In this context we stay compatible to the STRIPS definition of a plan.

2.2 Operators

Operators are the equivalent to actions used in a STRIPS planning domain and will be seen as processing steps. Operators consist of preconditions, a set of input data and an executional part (postconditions). An operator returns False if it is not applicable to the current task as determined by the preconditions, a new resource, a so called aggregate (combined from two or more resources of the input set) or one or many modified resources.
There are four types of operators to be included in our approach. All operators receive a set of resources as input parameters. We introduce the four operator types as follows:

1. *AttributeOperator*: Attribute Operators remove the given resources from the world state and for each resource add a new resource with an attributed name.
2. *MergingOperator*: Merging Operators combine two or more resources to generate a new resource, a so called aggregate. The newly generated resource loses all of the attributes the merging ingredients might have possessed.
3. *AddingOperator*: The Adding Operator is used when a certain resource is added to another resource while having the chance to remove it later on. The operators receives a list of resources and adds the first n-1 resources to the nth resource.
4. *RemovingOperator*: Removing operators remove resources which have been added by the adding operator.

2.3 Ontology

To implement preconditions we chose to rely on an ontology modeling dependencies between resources. The main part of the ontology is a taxonomy of resources. Is is created from an existing taxonomy that is enriched by eligibility erlations derived during a parsing process of existing recipes and their corresponding workflows. Using the information given by the By enriching this ontology using eligibility constraints, we can determine not only if an operator is eligible to be used with a certain set of resources, but moreover can we make use of the possibility to find variant resources easily. To achieve this we define a similarity measure on our given ontology which provides us with alternative resources.
Other advantages of using an ontology instead of modeling all required variants in a planning domain itself include, that we can reuse expert knowledge which has already been collected and verified.

2.4 Workflows

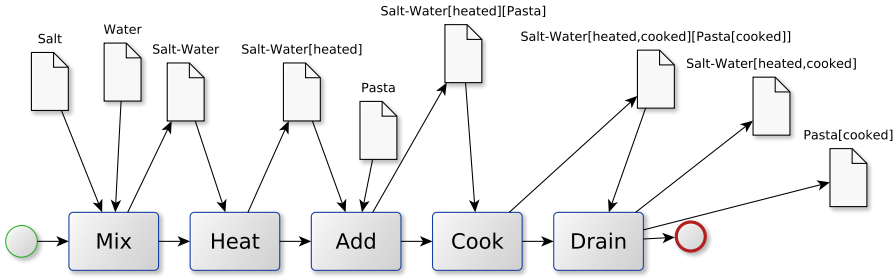

Fig. 1. Graphical representation of a simple workflow

Previously we presented the planning problem description and a description of its elements. Figure 1 provides a simple example of a workflow that corresponds with the planning problem description.
In this figure rounded boxes represent activities, input resources of activities are marked using an arrow to the activity and products of an activity are represented using an outgoing arrow from the activity. Activities in this workflow in fact conincide with the operators of the planning domain. We therefore will refer to them using the four operator types. Input resources and products can as well be treated as an equivalent to input resources and created resources in the planning domain.
The sample workflow describes the process of cooking pasta. At first the *MergingOperator mix* is used to merge the two resources water and salt to the new resource *salt-water*. Using the *AttributeOperator heat salt-water* gains the fact/attribute of heated.
In the next step pasta is being added to the *salt-water* by applying the *AddingOperator* add. We cannot use a *MergingOperator* at this point because the water could be used

for other purposes after the pasta has been cooked. By applying the *AttributeOperator* cooking on the salt-water with added pasta, both the added element and the salt-water receive the new fact cooked. After cooking, the pasta is drained, thereby separating pasta and salt-water from each other. This requires the use of the *RemovingOperator* drain. Finally, we receive two products: *salt-water[cooked]* and *pasta[cooked]* whereas *pasta[cooked]* is our anticipated result for this workflow. If we for this example without loss of generality assume, that all the operators of the planning domain have been used and are only eligible to the input resources being used in this example workflow, we can interpret the workflow as a valid plan to reach the goal of cooking pasta. The anticipated plan in our resulting plan format is shown in Listing 2:

```
(mix,{salt,water}),
(heat,{salt-water}),
(add,{pasta,salt-water[heated]}),
(cook,{salt-water[heated][Pasta]}),
(drain,{salt-water[heated][Pasta]})
```

Listing 2. Workflow Plan Format

In the given notation, a plan consists of a list of tuples of (operator,resourceset), whereas the resourceset consists of resources required for a valid execution of the given operator.

3 Planner

Our planner can be described as a semantic-aware planner and tries to determine all possible combinations of planning steps and constructs the corresponding plans matching the following criteria:

- Every operator can only be used once per resource
- Every resource can only be used until it is integrated in an aggregate or is removed for some other reason in the cooking process
- An ingredient is either available or unavailable
- The planner stops if no operator is applicable to any resource anymore or the maximum planning depth has been reached

Algorithm 1: Semantic-Aware Planning

```
   Data: ops: Set of operators
   Data: plans: Resulting Set of Plans
   Data: ontology : Eligibility relation between resources and ops
 1 Algorithm forwardPlanning(initialState,maxDepth)
       input  : initialState: the initial world state, i.e. a set of resources
                maxDepth: maximum recursion depth
       output: the resulting plans
 2     begin
 3         seekPlan(initialState,maxDepth);
 4         return plans;
 5 Function seekPlan(state,depth)
       input  : state:  current state of the world, i.e. a set of resources
                depth: recursion depth
       output: the resulting set of plans
 6     begin
 7         iPlans=∅;
 8         if depth==0 then
 9             return iPlans;
10         powerset=Powerset(state.resources);
11         forall the curRes in powerset do
12             forall the operator in ops do
13                 Boolean eligible=ontology.isEligible(curRes,operator);
14                 if eligible==True then
15                     tempState=update(state,curRes,operator);
16                     retPlans=seekPlan(copy(tempState),depth-1));
17                     forall the p in retPlans do
18                         iPlans=iPlans∪{operator ∘ p}
19                     plans=plans∪iPlans;
20         return iPlans;
```

Algorithm 1 works as follows: Beginning in the method *forwardPlanning* it receives the initial world state *initialState* and the maximum planning depth to consider. The initial state consists of the given resources and their status attributes. Furthermore the algorithm has access to the operators of the planning domain *ops*, the ontology used for determining eligibility and a result set of plans which is empty when the algorithm starts.

The algorithm continues by calling the actual planning subroutine *seekPlan*. This subroutine expects the current state of the world *state* and the current difference from the maximum planning depth and the current planning depth.

In the first step *seekPlan* constructs an empty set of intermediate plans for further use. A plan is hereby defined as an ordered list of tuples (operatorname,resources). It continues by checking if the maximum planning depth has been reached. If so it will return the

set of intermediate plans. The next step of the algorithm creates a *powerset* of all given resources in the current state. For all sets of resources in the powerset, the algorithm will determine if and which operators can be executed on the corresponding resource-sets by checking its eligibility. To check the eligibility of a resource and an operator, three checks are performed by *ontology.isEligible*:

1. Ontology Check: Here we determine if an operator/resourceset combination has been spotted in a realworld example so far and has therefore been entered in our ontology. An ontology check is true if and only if the resource is eligible by the ontology and all resources having temporarily been added to this resource using an AddingOperator are eligible to this method as well.
2. Was Applied Check: As a planning constraint we expect our resources to can be only executed with a given operator once. The planner records whether a resource has been applied already within a particular plan.
3. Availability Check: If a resource has been used in an *AddingOperator*, the resource is temporarily unavailable for further processing until it will be removed by a *RemovingOperator*. Unavailable resources can therefore not be applied to an operator.

If an eligible resource/operator pair has been found the corresponding operator will be executed, thereby updating the world state. We continue the algorithm by calling *seekPlan* recursively. It will receive a copy of the current world state and the depth parameter reduced by one. The recursive function returns either False if a planning step is not successful or one tuple per planning step, representing the called operator and its resourceset. Such intermediate plans that have been planned by the recursion call will be collected in *retPlans*. Next, *retPlans* will be merged into the set of intermediate plans by adding $operatorname \circ retPlan$ to it. The set of intermediate plans will then be integrated into the global resultset *plans*. When the recursion call comes to an end, the set of intermediate plans will be returned.

4 Application to Cooking

To apply this concept of planning on a real world domain, we have chosen the domain of pasta recipes. A representative sample of this domain was chosen by randomly selecting 30 recipes from an online cooking community [1] and manually searched for ingredients and operators used in those recipes. As myrecipes.com is a website where users are encouraged to post their own recipes we accepted those recipes as expert knowledge to be integrated in our domain.
In a first step we want to try to replan already existing recipes and in a later step producing variants using certain optimality criterias such as: Time, Concurrency, As less ingredients used as possible etc.

Example We will present a small example of a domain and a possible workflow creation in this section. All information our ontology consists of is assumed to have been inserted by extracting relationships from expert knowledge. Yet our ontology will be

[1] http://www.myrecipes.com[last visit: September 5, 2014]

radically simplified to only consist of water and coffee beans as resources and mix,cook and grind as operators. Our ontology is therefore defined as shown in Figure 2:

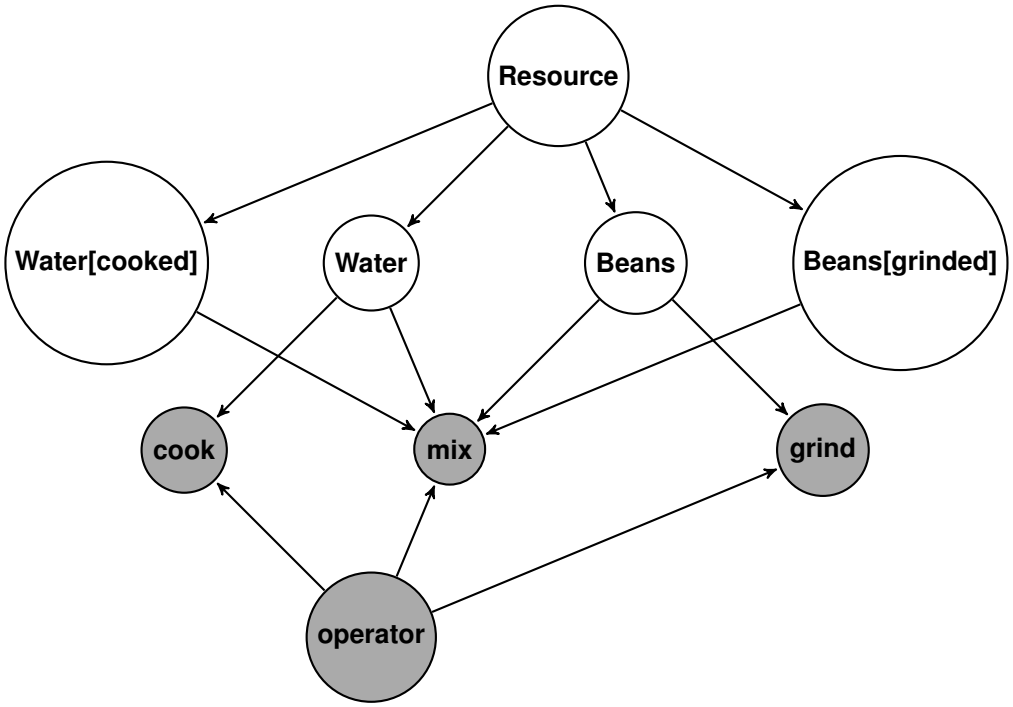

Fig. 2. Minimum Cooking Ontology

We hereby refer to operators as being gray and to ingredients in white. Cook and Grind can be classified as attribute operators whereas mix belongs to the class of merging operators. If an ingredient possesses a certain attribute it will be shown in brackets. If an ingredient is eligible to be used with an operator, they are connected.
In figure 3 we will present an example of the creation of a possible workflow that can be created as variants using the aforementioned basic ingredients using the proposed planning algorithm. Each box in the following picture will document a planning step of the algorithm.
In the first step we can see that the algorithm has several choices to choose from, as there are three eligible methods for at least one subset of the powerset of ingredients. The created plan is at this stage an empty list of tuples. After choosing the first planning operator, the *AttributeOperator grind* with the resource *beans*, the resource *beans* will get its appended attribute *grinded*. We therefore gain a modified set of resources in the next planning step. As the resource set is modified a recalculation of eligible methods is performed, resulting in removing the method grind(beans) from the set of eligible methods. The plan receives its first planning step *grind(beans)*.

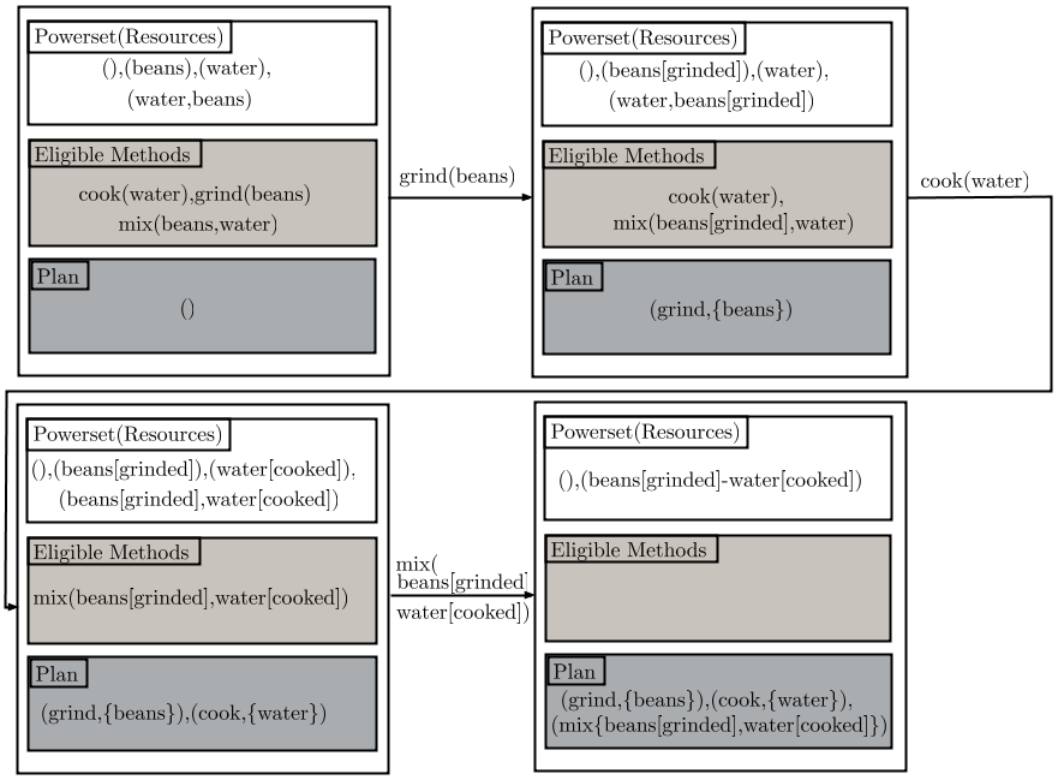

Fig. 3. Cooking Example

The second step repeats the procedure described for the operator grind with the operator *cook* and *water*, as *cook* is an *AttributeOperator* as well.
In the last step we can see that only one eligible operator is left to try, the *MergingOperator mix*. The algorithm will consequently apply this one next. The merging operator merges the resources *beans[grinded]* and *water[cooked]* together, leaving only the empty set and the resource *beans[grinded]-water[cooked]* in the powerset. Clearly at this point no eligible operator is left to be applied anymore, the algorithm will recognise the current plan as a valid solution and will return it.
The documented workflow might be the most intuitive one. However, our planner would as well output the variants shown in Listing 3 which are more or less useful.

```
cook(Water)-mix(Water[cooked],Beans)
cook(Water)-grind(Beans)-mix(Beans[grinded],Water[cooked])
grind(Beans)-mix(Water,Beans[grinded])
grind(Beans)-cook(Water)-mix(Beans[grinded],Water[cooked])
mix(Water,Beans)
```

Listing 3. Possible alternative plans

5 Related work

In this section we discuss some existing work on workflow planning. Masoumi et al. propose a planning based approach for the automated design of chemical processes. They formulated the problem as planning problem in Situation Calculus. In contrast to our planned approach their process planning method is strictly sequential and does not allow parallel or disjunctive control-flows.

Gil et al. developed a workflow generation and mapping system, that integrates an AI planning system into a grid environment. A desired data product is submitted by a user in an application-level description. The system generates a workflow by selecting appropriate application components and assigning the required computing resources. They aim at finding one optimal workflow to reach a goal while our approach aims at maximizing the number of workflows variants to reach a goal.

Klusch et al. [7] presented OWLS-Xplan, a planner for semantic web service composition planning. The web services are described in OWL-S which they convert to PDDL. In a second phase they used Xplan which is a hybrid planner that combines guided local search with graph planning and a simple form of hierarchical task networks to plan a sequence of web services. The approach to describe services or activities in OWLS and convert this description to a PDDL planning domain could be applied to our approach but OWLS is more complicated than our simple ontology format. The use of OWLS would produce a higher modeling effort which would conflict with our goal to reduce the human effort.

6 Conclusion and Future Work

We presented first steps to create workflow variants by means of planning. Our planning approach uses a semantic eligibility function which reduces the modeling effort for planning domain description. Our first prototypical implementation only supports a sequential control-flow. It is essential that a parallel or disjunctive control-flow is supported in future. The planning of workflow with a parallel control-flow is can be realized by the application of a partial-order planner. Partial-order planners do not plan strict sequences of actions but a plan is a set of action in combination with a partial ordering representing a ”before” relation on actions [8][p. 364]. This is similar to the meaning of AND-blocks. The order of activities in branches of an AND-block is only defined for activities in the same branch. There exists no ordering relation between activities of two different branches. In addition we intend to create a hierarchical task network that includes not only sequences of primitive action as implementation of a high-level action but also workflow snippets. As as result our approach could plan workflows with XOR- or LOOP-blocks.

7 Acknowledgments

This work was funded by the German Research Foundation, project number BE 1373/3-1.

References

1. Bergmann, R., Gil, Y.: Retrieval of semantic workfows with knowledge intensive similarity measures. In: Case-Based Reasoning. Research and Development, 19th International Conference on Case-Based Reasoning, ICCBR 2011. Volume 6880 of Lecture Notes in Computer Science., Springer (2011) 17–31 1
2. Müller, G., Bergmann, R.: A cluster-based approach to improve similarity-based retrieval for process-oriented case-based reasoning. In: 20th European Conference on Artificial Intelligence (ECAI 2014), IOS Press (2014) Preprint of this article available for download. 1
3. Montani, S., Leonardi, G.: A case-based approach to business process monitoring. Artificial Intelligence in Theory and Practice III **331/2010** (2010) 101–110 1
4. Minor, M., Bergmann, R., Görg, S.: Case-based adaptation of workflows. Information Systems **40** (March 2014) 142–152 1
5. Bergmann, Ralph, Minor, Mirjam, Islam, Siblee, Schumacher, Pol, Stromer, Alexander: Scaling similarity-based retrieval of semantic workflows. In Lamontagne, Luc, Recio-Garcia, Juan A., eds.: ICCBR-Workshop on Process-oriented Case-Based Reasoning, Lyon (2012) 15–24 1
6. Schumacher, P., Minor, M., Schulte-Zurhausen, E.: Extracting and enriching workflows from text. In: Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration. (2013) 285–292 1
7. Klusch, M., Gerber, A., Schmidt, M.: Semantic web service composition planning with owls-xplan. In: AAAI Fall Symposium on Semantic Web and Agents, USA. (2005) 5
8. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach (3. internat. ed.). Pearson Education (2010) 6