
http://www.puk-workshop.de

Jürgen Sauer¹, Stefan Edelkamp², Bernd Schattenberg³ (eds.)

1 Universität Oldenburg
Department für Informatik
D-26111 Oldenburg
juergen.sauer@uni-oldenburg.de

2 Universität Bremen
Technologie-Zentrum Informatik
D-28357 Bremen
edelkamp@tzi.de

3 Büro für intelligente Technologie-Beratung
Im Gehag 11
D-88471 Laupheim
E-Mail: mail@berndschattenberg.de

Stuttgart, September 2014
Program Committee

Stefan Edelkamp, University of Bremen, D
Lothar Hotz, University of Hamburg, D
Ulrich John, SIR Plan GmbH, Berlin, D
Thorsten Krebs, Encoway, Bremen, D
Sascha Ossowski, URJC, Madrid, E
Jürgen Sauer, University of Oldenburg, D
Bernd Schattenberg, Ulm, D
René Schumann, University of Sierre, CH

Organizers and Contact

Prof. Dr. Stefan Edelkamp
Technologie-Zentrum Informatik und Informationstechnik
Universität Bremen
e-mail: edelkamp@tzi.de

Prof. Dr.-Ing. Jürgen Sauer
Universität Oldenburg
Fak II, Department Informatik
e-mail: juergen.sauer@uni-oldenburg.de

Dr. Bernd Schattenberg
e-mail: mail@berndschattenberg.de
<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Challenges in Key Performance Indicator Based Planning and Control in Multi-Agent Systems</td>
<td>Florian Pantke</td>
</tr>
<tr>
<td>Towards workflow planning based on semantic eligibility</td>
<td>Timo Homburg, Pol Schumacher and Mirjam Minor</td>
</tr>
<tr>
<td>Social acceptability of opportunistic behaviour of an assistant robot in human–robot everyday collaboration</td>
<td>Nataliia Kushnirenko, Alexandra Kirsch</td>
</tr>
<tr>
<td>Finding User-friendly Linearizations of Partially Ordered Plans</td>
<td>Daniel Höll, Pascal Bercher, Felix Richter, Marvin Schiller, Thomas Geier, Susanne Biundo</td>
</tr>
<tr>
<td>Introducing Hierarchy to Non-Hierarchical Planning Models -- A Case Study for Behavioral Adversary Models</td>
<td>Louisa Pragst, Felix Richter, Pascal Bercher, Bernd Schattenberg, Susanne Biundo</td>
</tr>
<tr>
<td>Design and Redesign of University Course Programs</td>
<td>Ulrich John</td>
</tr>
<tr>
<td>Multi Goal Motion Planning</td>
<td>Stefan Edelkamp</td>
</tr>
</tbody>
</table>
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 expe-
rience. 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 prod-
ucts 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 work-
flows. 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
resources anymore. The operators are divided in four classes, as described in the sub-
section 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:

\[
\text{Have}(\text{water}) \xrightarrow{\text{heat}} \neg \text{Have}(\text{water}) \land \text{Have}(\text{water})[\text{heated}]
\]

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 re-
course and possible other facts worth noting during the planning process.
Problem Description 1: Planning Problem

<table>
<thead>
<tr>
<th>Initial state:</th>
<th>{Have(r) : r available resource}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal state:</td>
<td>∀r, o: Have(r) : (¬Eligible(r, o) ∧ Have(r))</td>
</tr>
</tbody>
</table>

1 //ActionTypes: 4 different operator types as explained below
2 Rcurr = {r : r resource on which operator is applied}
3 AttributeOperator (o(Rcurr)),
4 PRECOND: ∀r ∈ Rcurr : Have(r) ∧ Eligible(r, o)
5 EFFECT: ∀r ∈ Rcurr : ¬Have(r) ∧ ∀r ∈ Rcurr : Have(r[o])
6 MergingOperator (o(Rcurr)),
7 PRECOND: ∀r ∈ Rcurr : Have(r) ∧ Eligible(r, o)
8 EFFECT: ∀r ∈ Rcurr : ¬Have(r) ∧ Have(rnew)
9 AddingOperator ((o(Rcurr)),
10 PRECOND: ∀r ∈ Rcurr : Have(r) ∧ Eligible(r, o)
11 EFFECT: ¬Have(r1) ... ¬Have(rn) ∧ Have(rn[r_1 .. r_{n-1}])
12 RemovingOperator ((o(Rcurr)),
13 PRECOND: ∀r ∈ Rcurr : Have(r) ∧ Eligible(r, o)
14 EFFECT: ¬Have(rn[r_1 .. r_{n-1}]) ∧ Have(r1) ... Have(rn))

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 relations 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

![Graphical representation of a simple workflow](image)

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 coincide 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 \texttt{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 \texttt{RemovingOperator}
drain. Finally, we receive two products: \texttt{salt-water[cooked]} and \texttt{pasta[cooked]} whereas
\texttt{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:

\begin{verbatim}
(mix, {salt, water}),
(heat, {salt−water}),
(add, {pasta, salt−water[heated]}),
(cook, {salt−water[heated][Pasta]}),
(drain, {salt−water[heated][Pasta]})
\end{verbatim}

\textbf{Listing 2. Workflow Plan Format}

In the given notation, a plan consists of a list of tuples of (operator,resource set), whereas
the resource set 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 pos-
sible 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 max-
  imum 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
   begin
   seekPlan(initialState,maxDepth);
   return plans;

2 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
   begin
   iPlans=∅;
   if depth=0 then
   return iPlans;
   powerset=Powerset(state.resources);
   forall the curRes in powerset do
   forall the operator in ops do
   Boolean eligible=ontology.isEligible(curRes,operator);
   if eligible==True then
   tempState=update(state,curRes,operator);
   retPlans=seekPlan(copy(tempState),depth-1);
   forall the p in retPlans do
   iPlans=iPlans∪{operator ∘ p}
   plans=plans∪iPlans;
   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 \texttt{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 \texttt{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 \texttt{retPlans}. Next, \texttt{retPlans} will be merged into the set of intermediate plans by adding \texttt{operatorname} \texttt{retPlan} to it. The set of intermediate plans will then be integrated into the global resultset \texttt{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 \footnote{http://www.myrecipes.com\[last visit: September 5, 2014\]} 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
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:

![Minimum Cooking Ontology](image)

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.

Listing 3. Possible alternative plans

```
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 )
```
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. \cite{Klusch2006} 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 \cite[p. 364]{Birke2001}. 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


Social acceptability of opportunistic behaviour of an assistant robot in human–robot everyday collaboration

Nataliia Kushnireuko, Alexandra Kirsch

Department of Computer Science
University of Tuebingen
Sand 14
72076 Tuebingen
natalia.kushnirenko@uni-tuebingen.de
alexandra.kirsch@uni-tuebingen.de

Abstract: In dynamically changing environment, an autonomous robot and human, performing a joint task together, should take decisions depending on many factors. A fundamental feature of human decision–taking is opportunism, and in this paper we explore how socially acceptable is opportunistic behaviour of robot in every day life situation with human interaction. We conducted a pilot study to assess whether a robot using opportunities is regarded as a better companion. Showing videos of a simulated robot, we asked for personal evaluations of subjective factors as helpfulness, politeness and competency.

1 Introduction

Making autonomous robots act understandably for humans is an important prerequisite for the human–robot collaboration in everyday life. We are interested in the design and implementation of flexible planning and plan execution strategies for robots. One aspect that characterizes human activity is that actions are not all planned in advance, but that humans react to opportunities for efficient action selection. For example, when we tidy up a room, we store away the things as we see them lying around, rather than making a list of all things to be tidied up and then storing the objects respectively. According to Botvinick [Bot08] there is a “[...] research that suggests the existence of two systems for action control: a habit system based on context–response associations and a goal–directed system operating through the anticipation of action outcomes.” This means that it would be unnatural for an assistant robot to suggest a complete plan to a person to execute jointly, because humans decide their next action from the perceptual context rather than a premeditated plan [SWB97].

In the context of our work on a robot assistant we are specifically focusing on opportunities that appear from dynamics caused by humans and how to detect and exploit them to behave
in a legible ¹, human–friendly way.

It is very difficult to evaluate robot behaviour efficiently with regard to user acceptance. The very nature of opportunities requires a variety of situations with different kinds of opportunities arising in different contexts of task execution. Also, one might argue that a robot does not need to be opportunistic, but it should rather be predictable [Nor07]. Thus, before investing time in a specific strategy, we conducted a pilot study to assess whether a robot with some opportunistic behaviour is better accepted than one following a strict plan.

We measure the performance of opportunistic planning along the social acceptability dimension, which includes objective measures such as waiting effort of the human during the task, the qualities of the robot being safe, pleasant, friendly, polite, competent, knowledgeable, responsible, intelligent, efficient, helpful, able to predict human intention, and the level of social willingness to buy this kind of robot. This self–assessment is complemented by open questions.

We present a video–based study in which the participants watch a scene without being occupied with a task, and rate the video according to proposed criteria.

2 Background and related work

Kruse and Kirsch [KK10] have proposed an architecture for opportunistic action selection as shown in Figure 1. A parallel process is monitoring the plan execution, generates possible actions and evaluates them according to a given model. In case of positive evaluation result an opportunity filter is checking if a change to these alternative actions would bring predicted benefit according to the model. In our study, we use this architecture and examine possible opportunity models. The used architecture builds on concepts from goal–driven (or goal–directed) autonomy [MKA10] with its most prominent architecture, the Belief–Desire–Intention (BDI) framework [Bra87, GPP+98]. Similar to behaviour–based approaches an agent has abstract desires, which can be compared to motivations of Sevin and Thalmann [dST05], but more often represent goals [PH99, MKB04]. A desire can become an intention when the agent tries to achieve a specific goal and corresponds to a plan to achieve this goal.

Figure 1: Detecting and using opportunities.

¹By legibility we describe the ability of humans to understand the intentions of the robot intuitively by observation of its behaviour, which makes it possible to anticipate the robot’s next actions.
Under the term “intention reconsideration” Schut and Wooldridge [SW01] have described a general framework in the BDI model, in which the reconsideration of the current goal is treated like a physical action. In their framework, the cost of a reconsideration action is compared to the benefit that could be obtained by changing the course of action. In this way, an agent only reconsiders its intended course of action if it believes that it can significantly improve its performance.

Reconsideration strategies have also been explored by Pollack and Ringuette [PR90]. They introduced the notions of bold and cautious agents. They call an agent cautious when the agent is “very sensitive to its environment, willing to reconsider its plans in response to a wide range of events”. The same holds for opportunistic agents, the difference being that a cautious agent reconsiders to avoid making mistakes, an opportunistic agent reconsiders to be more acceptable and possibly more efficient.

Other examples for plan reconsideration in robotics are given by Helwig and Haddawy [HH96] for a truck–world simulation, or by Miura and Shirai [MS02] for an office robot. All these approaches work on operators defined by first–order predicates. But opportunities are often more interesting when considering more specific knowledge. This is taken into account by Parsons et al. [PPSW00] who used feedback from a fuzzy controller to recognise opportunities in the framework of Schut and Wooldridge [SW01].

Broz, Nourbakhsh, and Simmons [BNS08] consider the adequate timing of robot actions with respect to humans using POMDPs. Cirillo, Karlsson, and Saffiotti [CKS08] describe a planning framework in which a robot takes into account the activity of nearby humans and adapts its actions to not disturb or distress humans. This approach considers human comfort by avoiding the human, rather than contributing to a joint goal. In contrast, legibility in joint tasks is an explicit goal of the Human–Aware Task Planner by Alami et al. [RAC06]. They use an HTN planner to generate plans not only for the robot, but also for a human partner. Legibility is modelled as a cost function for plans. This planner is integrated into a classical three layer architecture [ACF+98] and thus cannot deal directly with the dynamics of the world. A new plan is only generated when the current one fails. This approach assumes that the robot can plan for itself and the human and that the human will accept this plan.

In other contexts, there are approaches that explicitly take into account the movement of humans [ZRG+09] or even the intentions of humans [CvSK+06, BTJB10]. Benson and Nilsson [BN95] use a more general model for opportunistic behaviour. Their architecture includes an inherent trade–off between the priority of an action and the estimated time to achieve it.

The models mentioned so far all assumed that the information about actions is available by definition [BN95, dST05], which is unrealistic in everyday environments, or is provided by a lower–level model that also achieves the given navigation goals [PPSW00, KK10], which might entail problems for execution efficiency.

To our knowledge, the acceptance of opportunistic behaviour has never been shown.
3 Method

We are specifically interested in using opportunistic methods in everyday real–world environment for a robot that collaborates with humans. Therefore a collaborative table–setting scenario was chosen for testing social acceptability.

To this end we have conducted a user study to evaluate the social acceptability of robot behaviour with a set of videos recorded with the Morse simulator, in which a human and a robot are setting the table together. The Morse simulator [LGK^+12] is a project of LAAS–CNRS to supply researchers with a realistic, physical simulator for a wide variety of robot tasks. This simulator is based on the free 3D modelling software Blender and allows us to include an animated human that can move in the world and manipulate objects. We use the PR2 robot for demonstrating opportunistic planning. The basic functionality is provided by open–source ROS modules. Both the robot and the human can be controlled with the keyboard as in a computer game. For the recording the videos for our study the robot was controlled by a control script, while human avatar was moved manually.

The kitchen environment where the robot is bringing the plate and the human is holding the spoon is shown in Figure 2.

![Figure 2: Robot and human performing the tasks in our kitchen environment.](image)

The user study was not held as Wizard of OZ experiment, because the task of controlling human avatar might both distract user and increase the number of possible reasons of certain robot behaviour observed. Moreover, in this case each participant would have different experience. A real robot would be even more complicated object to judge, therefore, keeping experiment fixed should allow us to figure out which exactly kind of robot behaviour would be more acceptable.

The videos for the user study were generated considering different combinations of opportunity models[^3]. We examined the following models: 1) spatial comfort for the human

[^2]: http://morse.openrobots.org

[^3]: These models were taken into account for constructing the scenes. The robot did not autonomously decide, but was scripted.
when the robot moves to an object; 2) the time to grasp an object, taking into account if cupboard doors are currently open or closed. In addition, we were interested in the degree of collaboration and mutual considerateness.

3.1 Participants

33 participants of the age 21–40 years with the average age of 28 years took part in our experiment — thereof 12 women and 21 men. 30 of them did not have experience with robots. The number of people taking the test simultaneously varied from one to four.

3.2 Procedure

After personal data collection and short introduction with the general task description (both printed and read aloud by the experimenter), users were asked to watch the videos in random order and to answer a set of questions on paper after each video. The experimenter’s role was mainly to observe the users watching the videos.

Since the movements in Morse are rather slow, all videos were sped up to 218%. In a pilot study with three participants the normal speed of the movements was used and those results were comparable to those of the main study. This shows that the video speed does not seem to affect the users’ opinions.

The videos can be seen at http://vimeo.com/hcai/videos. The first video, lasting for 2:33 minutes, was used as an introduction to make observers used to the robot in the simulated environment.

All videos include scripted behaviour of the robot except the last video with the shifting objects scenario, where the robot was controlled manually. Every video represents a certain situation from the real world.

3.3 Videos description

Each of the offered videos is based on one of scenarios, summarized in Table 1 with parameters. Opportunity is an action the robot can perform alternatively or following the current task considering the dynamics of the environment, such as picking a different object, which will be useful later rather than waiting for the free space to reach the object according to the current plan; or using a chance to take the item from an opened drawer, as the reachability of this item might become worse later.

The videos with opportunities used include situations when the man is blocking the way, and the robot has changed the original plan in favour of an alternative action, such as picking the object from the other side of the table; using a chance to take the object from the opened drawer or pick another item to avoid waiting.
Table 1: Video scenario characteristics

<table>
<thead>
<tr>
<th></th>
<th>Opportunity used</th>
<th>Collaboration</th>
<th>Robot waits</th>
<th>Human waits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video 1 (Introduction)</td>
<td>yes</td>
<td>together</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Video 2 (Human, then robot)</td>
<td>no</td>
<td>one by one</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Video 3 (Human waits)</td>
<td>no</td>
<td>together</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Video 4 (Two cups)</td>
<td>yes</td>
<td>together</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Video 5 (Robot waits)</td>
<td>no</td>
<td>together</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Video 6 (Opened door)</td>
<td>yes</td>
<td>together</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Video 7 (Shifting)</td>
<td>no</td>
<td>one by one</td>
<td>yes</td>
<td>-</td>
</tr>
</tbody>
</table>

Collaboration type can be mutually dependent ("together"), as the robot and human are moving and performing the tasks simultaneously, or almost independent ("one by one"), when the robot only acts during the time the human is far away or inactive.

We are using “robot waits” and “human waits” as parameters to indicate when pure waiting time of robot or human respectively was obvious.

Both qualities safe and polite are combined into one column in Table 1, as according to our imagination, in presented situations the robot is either both safe and polite, or unsafe and impolite.

3.4 Scenario

The main scenario of the videos is a man who brings the knife, the fork and the spoon to the table, while the robot is in charge of bringing the plate, milk and cup to the same table in the kitchen. We tried different variations of this scenario, so that situations when robot and man are completing the task one by one, with and without collision, with the opportunity used, with the opportunity used due to the human blocking the original way, with robot ignoring the human.

Let us describe the offered videos related to the opportunities provided:

Video 1:

The introduction video shows human and robot setting the table together one by one without having any conflicts till some point, when the human is blocking the way for the robot to reach the table, and the robot starts slightly shifting human to the side. As soon as the human leaves the way free, the robot continues completing the task of taking the blue cup, but chooses a new opportunity of taking it from the other side of the table. The results of this video are separated and are not included in evaluation.

Video 2:

First the human brings the fork, the knife and the spoon, and only after the human
stops the robot brings the plate, milk and cup. This scenario does not contain opportunities, because robot’s behaviour can be repeated with a usual plan if there is no human on the way.

Video 3:
Whenever there is a collision on the way, the human is waiting for the robot to finish its task. The robot does not use any opportunity, simply following the plan.

Video 4:
When the human is occupying the place near the table, the robot uses an opportunity to take the cup from another place with help of both hands to take two cups.

Video 5:
Whenever a collision occurs, the robot waits until the human frees the space and never uses any opportunity.

Video 6:
The human forgets to close the drawer, and the robot uses this opportunity to take the bowl from it. The human closes the drawer afterwards.

Video 7:
The human is bringing items to the table, and the robot is shifting them organizing more space while the human is far from the table. The robot uses opportunities to work around the table while the space is free from human.

3.5 Questionnaire

In order to evaluate social acceptability of the robot, the questions were created partially on the basis of the Godspeed test [BKCZ09]. The users were asked to choose the number from 1 (the best) to 5 (the worst) on a semantic differential scale to rate the selected qualities of the robot shown in each video: Safe–Unsafe, Pleasant–Unpleasant, Friendly–Unfriendly, Polite–Impolite, Competent–Incompetent, Knowledgeable–Ignorant, Responsible–Irresponsible, Intelligent–Unintelligent, Efficient–Inefficient, Helpful–Unhelpful. These items were chosen in order to keep the questionnaire clear, and to allow the user to describe the robot behaviour feedback also in other interesting aspects for better evaluation.

Despite some of these items seem to be closely related, we selected them for user rating to be sure in the final results [BM10]. These measurements were supplemented with a set of open questions, namely:

1. Please describe the strategy: did you notice any pattern of how the robot and human interacted?
2. Did you notice particularly bad or annoying robot behaviour?
3. Did you notice particularly good robot behaviour?
4. Where you surprised at some point? (What were you expecting instead?)

5. Does the robot predict human movement? (Yes/No)

The question about surprise is an additional subjective criterion to evaluate other aspects of the robot behaviour, helping to show if the robot’s action and goal can be inferred correctly. The last question was asking to put the overall rating for the video in the points from 1 to 5 expressing the willingness to buy the robot behaving as in the shown video. After completing the survey for all videos, the participants were asked to respond whether the robot or a human was faster, and to provide any other comments.

4 Results and evaluation

The results of the user ratings are presented in Figures 3 and 4 showing the chosen values from 1 to 5 for the certain quality in the group of videos with similar parameter according to Table 1. The smaller number corresponds to more positive rank, e.g. “1” is safe and “5” is unsafe. The biggest difference in the results is seen for the groups of videos related to collaboration: together or one by one, and related to the opportunity usage: with or without opportunities. Both figures show, that there is only slight change in the ratings for “polite” quality, in the one-by-one collaboration scenario the robot is safer, more competent, helpful and the will to buy it is also higher. The user ratings of the same robot qualities for the opportunity usage video group barely change.

![Figure 3: Collaboration dependent qualities rates](image)

The robot only acting while the human inactive has the highest average ranks (2.6 for "I would buy it"), as well as the robot which has to wait for the human to free the space needed to complete the task (2.9 for "I would buy this robot").

As expected, the situation when the human has to wait for the robot is the least pleasant (average rank 3) and safe (2.45). Also, the least safe and pleasant the robot seems, the less will to buy the robot exists, despite the other qualities were rated pretty high.
The robot is evaluated to be less helpful when it makes the human wait. The scenario with shifting items on the table (video 7) shows the robot behaviour becomes efficient, but does not make people want to buy the robot.

During the user study participants had small doubts such as wonder why did not robot close the drawer in the video 6, or why would one human need two cups in video 4. Participants thought the robot remembers how many people live in the apartment or is perfectly aware of the human habits and therefore predicts his intentions. A couple of users were sure the robot repeats after the human trying to help, learning from the environment right away. The scores for the opportunistic behaviour are slightly better in safety dimension, but less in helpfulness in the scenarios with the opportunities used.

Interesting comments from the study came from three users who were sure there was verbal communication between the human and the robot, despite the videos being silent. Two users thought there is a remote control for the robot. Some people proposed to avoid collision situations with robot by separating the workspace or giving a certain task to the robot based on how heavy the object is.

Some participants mentioned not only the human safety, but also the safety of objects placed: in their opinion, video 7 demonstrates how careful the robot is to keep the items far from the edge of the table, or to find a proper place for them.

We used only some variables in the previous evaluation, because the others are highly correlated, which can be seen from Table 2. It also shows how independent are the qualities intelligent or efficient from friendly, pleasant, polite or safe. The coefficient which is 0.5 or higher is considered as strong correlation [PS08], and is highlighted.

All participants claim the speed of the robot was slow, but this fact brings contradicting thoughts: slow is good for safety, but sometimes not efficient enough.

The answers on the question if the robot predict human stay on the same shape of the lines presented in Table 3. Also our study shows the more robot predicts human behaviour, the more interest in buying the robot is present.
5 Conclusion

The question should the robot actually change the behaviour due to dynamics of the environment, or to do the first action intended was raised. We examined different scenarios with and without opportunity usage and asked users to rate them with respect to efficiency, friendliness, usefulness and legibility of the resulting behaviour. The spatial comfort and waiting effort of the human were taken into consideration to analyse the applicability to real-world environment.

To conclude, with our results in the social acceptability of opportunistic behaviour of an assistant robot in human-robot everyday collaboration, we could not support the finding of Norman saying that a robot does not need to be opportunistic, but it should rather be predictable [Nor07]. There is also no reliable indication that opportunism is better accepted, it is possible due to the small number of videos used, and due to existence of many other differences in scenarios, than just using or not using opportunities, which is also not easy to show clearly. The other distracting factors, such as robot movement, the task performed and various little details and changes influence user opinions more than opportunity-taking.

Table 2: Pearson correlation coefficient for all videos

<table>
<thead>
<tr>
<th></th>
<th>Pleasant</th>
<th>Friendly</th>
<th>Polite</th>
<th>Competent</th>
<th>Knowledgeable</th>
<th>Responsible</th>
<th>Intelligent</th>
<th>Efficient</th>
<th>Helpful</th>
<th>I would buy it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pleasant</td>
<td>0.55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friendly</td>
<td>0.46</td>
<td>0.69</td>
<td></td>
<td>0.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polite</td>
<td>0.51</td>
<td>0.59</td>
<td>0.76</td>
<td>0.51</td>
<td>0.68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Competent</td>
<td>0.43</td>
<td>0.49</td>
<td>0.4</td>
<td>0.37</td>
<td>0.73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knowledgeable</td>
<td>0.4</td>
<td>0.46</td>
<td>0.4</td>
<td>0.46</td>
<td>0.68</td>
<td>0.52</td>
<td>0.48</td>
<td>0.46</td>
<td>0.49</td>
<td></td>
</tr>
<tr>
<td>Responsible</td>
<td>0.56</td>
<td>0.49</td>
<td>0.53</td>
<td>0.6</td>
<td>0.59</td>
<td>0.68</td>
<td>0.52</td>
<td>0.55</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>Intelligent</td>
<td>0.26</td>
<td>0.34</td>
<td>0.26</td>
<td>0.2</td>
<td>0.55</td>
<td>0.6</td>
<td>0.55</td>
<td>0.51</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>Efficient</td>
<td>0.28</td>
<td>0.36</td>
<td>0.32</td>
<td>0.34</td>
<td>0.53</td>
<td>0.5</td>
<td>0.51</td>
<td>0.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Helpful</td>
<td>0.39</td>
<td>0.47</td>
<td>0.46</td>
<td>0.45</td>
<td>0.63</td>
<td>0.6</td>
<td>0.55</td>
<td>0.52</td>
<td>0.48</td>
<td>0.46</td>
</tr>
<tr>
<td>I would buy it</td>
<td>0.38</td>
<td>0.42</td>
<td>0.4</td>
<td>0.39</td>
<td>0.49</td>
<td>0.46</td>
<td>0.52</td>
<td>0.48</td>
<td>0.46</td>
<td>0.49</td>
</tr>
</tbody>
</table>

Table 3: User opinion on whether robot predicts human in each video.

<table>
<thead>
<tr>
<th>Video</th>
<th>“Yes” for robot predicts human, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video 1 (Introduction)</td>
<td>50.00</td>
</tr>
<tr>
<td>Video 2 (Human then robot)</td>
<td>44.44</td>
</tr>
<tr>
<td>Video 3 (Human waits)</td>
<td>55.56</td>
</tr>
<tr>
<td>Video 4 (Two cups)</td>
<td>50.00</td>
</tr>
<tr>
<td>Video 5 (Robot waits)</td>
<td>75.86</td>
</tr>
<tr>
<td>Video 6 (Opened door)</td>
<td>53.57</td>
</tr>
<tr>
<td>Video 7 (Shifting)</td>
<td>53.33</td>
</tr>
</tbody>
</table>
Furthermore, our results show how hard evaluation of autonomous behaviour is, as the real robot is even more uncontrolled than the robot with scripted behaviour we used basing on proposed model. We should be careful in setting up a truly realistic task, keeping in mind how important all the details are for user opinion, and to show as many interactions as possible. It would be also good to use larger semantic differential scale to be able to receive more precise answers for evaluation. For future studies objective measurements such as waiting times, time to reach the goal will be added.

We should mention that every person would behave differently in proposed situation. This research was using a passive study where users could only watch the videos. In future, the interactive model where users can control the human avatar in the simulated environment will be used for evaluation.

References


Finding User-friendly Linearizations
of Partially Ordered Plans

Daniel Höller, Pascal Bercher, Felix Richter, Marvin Schiller, Thomas Geier,
and Susanne Biundo
Institute of Artificial Intelligence, Ulm University, Germany
{forename.surname}@uni-ulm.de

Abstract. Planning models usually do not discriminate between different possible execution orders of the actions within a plan, as long as the sequence remains executable. As the formal planning problem is an abstraction of the real world, it can very well occur that one linearization is more favorable than the other for reasons not captured by the planning model — in particular if actions are performed by a human. Post-hoc linearization of plans is thus a way to improve the quality of a plan enactment. The cost of this transformation decouples from the planning process, and it allows to incorporate knowledge that cannot be expressed within the limitations of a certain planning formalism. In this paper we discuss the idea of finding useful plan linearizations within the formalism of hybrid planning (although the basic ideas are applicable to a broader class of planning models). We propose three concrete models for plan linearization, discuss their ramifications using the application domain of automated user-assistance, and sketch out ways how to empirically validate the assumptions underlying these user-centric models.

Keywords: Hybrid Planning, POCL Planning, HTN Planning, Plan Linearization, User-centered Planning, User Assistance, Plan Execution

1 Introduction

In the last years we showed in different application domains that AI Planning can help to give a human user support on complex tasks like the operation of a smart phone, where the system helps the user to send messages, create contacts or appointments [2]. Recently we described a system that helps the user assemble a complex home theater system [1]. A hybrid planning system is utilized to find plans that help the user to reach her or his goal. Hybrid planning combines elements from Hierarchical Task Network (HTN) planning and Partial Order Causal Link (POCL) planning. Abstract tasks are decomposed repeatedly until all tasks are specific enough to be applied directly. Causal links make the causal structure of the plan explicit and ensure executability. The plans are partially ordered and thus enable a flexible execution order. Any of these orderings will transfer the initial state to a state that fulfills the desired goal properties.

However, when the plans have to be communicated to a human user, or are even executed by a human, some linearizations of the partially ordered plan
might be better than others. We identified the following possible objectives for the task of plan linearization. The choice of the objective especially influences the experimental setup when the different utility functions are evaluated.

1. imitate human behavior
2. maximize the user’s subjective appraisal
3. optimize some objective metric – this could be, e.g., the time a user needs to execute a plan

We do not want to imitate humans’ behavior. In our context, the second and third objective are more relevant. Which of them to choose might depend on a specific application. In some situations, the user’s appraisal is the top goal. In others a speedy execution of the plan might be the ultimate objective.

In this paper we introduce and discuss three utility functions, relying on different properties of the planning problem and its solution, that enable the comparison of different linearizations. These utilities are discussed in context of our home theater assistance system [1].

The following sections introduce the hybrid planning approach (Sec. 2) and our example domain (Sec. 3). Sec. 4 introduces the different utility functions that are discussed in Sec. 5. That section also outlines several alternatives for empirically evaluating the proposed utility functions. Sec. 6 concludes the paper.

2 The Hybrid Planning Framework

Hybrid planning [1,3] combines elements from Hierarchical Task Network (HTN) planning and Partial Order Causal Link (POCL) planning. Like in HTN planning, abstract tasks are repeatedly decomposed using decomposition methods until they can be executed directly. Like in POCL planning, the system utilizes Causal Links to prevent already established preconditions from being changed. The domain modeler is free to allow the system to insert primitive tasks independent from hierarchical decomposition.

A planning domain is a tuple \( D = (T, M) \), where \( T \) is a set of tasks that can be partitioned into the sets \( T_C \) and \( T_P \) of compound and primitive tasks, respectively. It holds \( T = T_C \cup T_P \) and \( T_C \cap T_P = \emptyset \). Each \( t(\varpi) \in T \) is a tuple \( (\text{pre}, \text{eff}) \), where \( \text{pre} \) and \( \text{eff} \) are conjunctions of literals over the task parameters \( \varpi = \varpi_1, \ldots, \varpi_n \) and specify the preconditions and effects of a task. A partial plan has the structure \( P = (PS, \prec, VC, CL) \), where \( PS \) is a set of plan steps, i.e. of uniquely identified tasks. \( \prec \) is a set of constraints of the form \((t_i, t_j)\) that define a strict partial order on the tasks in \( PS \). \( VC \) is a set of variable constraints that codeignite or non-codeignite task parameters to other task parameters or to constants. \( CL \) is a set of causal links. A causal link has the form \( t \rightarrow \varphi \; t' \) and states that the precondition \( \varphi \) of task \( t' \) is established by task \( t \). The precondition \( \varphi \) is said to be supported by that causal link.

As given in the beginning, the compound tasks of a plan have to be decomposed until only primitive tasks are left. This is done using (decomposition) methods given by the second domain element \( M \). A method \( m \in M \) has the form
\[ m = (t(\tau), P) \] and defines a mapping of the compound task \( t(\tau) \) to a partial plan \( P \).

A planning problem \( P \) is given by a planning domain and a definition of an initial state, a goal state and an initial partial plan \( P_{init} \). Initial and goal state are usually given as two special primitive tasks \( init \) and \( goal \) that have the initial state as effect and the goal state as precondition, respectively. States and the applicability of primitive tasks are defined as usual. Fully grounded primitive tasks are also called \( actions \). A partial plan \( P = (PS, \prec, VC, CL) \) is a solution to a planning problem if and only if the following conditions hold:

1. It is a refinement of the initial plan \( P_{init} \), i.e. it is achieved by decomposition, insertion of causal links and ordering constraints, and (if intended by the domain designer) task insertion.
2. It does not contain compound tasks.
3. All preconditions are supported by causal links.
4. There is no causal threat. A threat is a situation where a task \( t' \) that has an effect \( \neg \varphi \) can be placed between two tasks \( t \) and \( t'' \) when there is a causal link \( t \rightarrow_{\varphi} t'' \) in \( CL \). A task \( t' \) can be placed between the tasks \( t \) and \( t'' \) when \( \prec \cup \{(t, t'), (t', t'')\} \) is also a valid strict partial order.

A partial plan that is a solution is also referred to as a \textit{plan}. We denote it by \( P^* = (PS^*, \prec^*, VC^*, CL^*) \).

### 3 Example Domain

We exemplify the proposed utility functions in the home theater domain \([1, Section "Domain Model"]\). In that domain several devices, such as a blu-ray player, a satellite receiver, an amplifier (audio/video receiver), and a television, have to be connected with each other, s.t. the television receives the audio signals of the blu-ray player and the satellite receiver and the television receives the video signals of these devices. Many different cables are modeled, such as cinch or DVI and HDMI cables. The specific devices and cables are modeled by constants of the respective sort.

For connecting these devices using cables, the domain features \textit{plugIn} actions that take as argument four constants: the two hardware components that have to be connected with each other (where one is a cable and the other a device such as a blu-ray player) and the two ports that are connected (for example, an HDMI port in case of a blu-ray player and one of the ends of an HDMI cable). Thus, for connecting two devices using one single cable, (at least) two actions are needed: one for each end of the cable. The domain is modeled in such a way that actions may only be executed in the order in which the signal is transported from its source to its destination. So, if a cinch cable were utilized to transport the audio signal of the blu-ray player to the amplifier, then the cable needs first to be plugged into the blu-ray player and afterwards into the amplifier although the pure "physics" would also allow the other execution order.
As an example, consider the following solution to the given planning problem:

The Blu-ray player is connected to the amplifier using two cables: a DVI cable is used for video and a cinch cable for the audio signal. The satellite receiver is connected to the amplifier using a scart-to-cinch cable. That cable transports both audio and video. It uses a scart connector at one end (that is plugged into the satellite receiver) and three single cinch ports at the other: two transport the audio signal and one the video. Finally, the amplifier is connected to the television using a further cinch cable. Since putting an end of a cable into a device is modeled using a single action, that solution consists of 10 plan steps (two actions for each cable except scart-2-cinch, which requires four). This solution is depicted graphically in Fig. 1.

Fig. 1: Fig. (a) gives a graphical representation of the solution to the described problem of the home theater domain. The graph in Fig. (b) schematically depicts a solution plan to the corresponding planning problem. Nodes $ps_i$ represent plan steps. There is an edge between two nodes if and only if there is at least one causal link between the corresponding nodes in the graph. The plan steps $ps_1$ and $ps_2$ represent the $\text{plugIn}$ actions to connect the Blu-ray player to the amplifier using a DVI cable, whereas $ps_3$ and $ps_4$ connect those devices using a cinch cable. The plan steps $ps_5$ to $ps_8$ represent the $\text{plugIn}$ actions for connecting the satellite receiver with the amplifier. Audio connection is established using $ps_7$ to $ps_8$ (left and right channel) and video using $ps_6$. The amplifier is connected to the television with the cinch cable using $ps_9$ and $ps_{10}$. Since cables may only be connected in the order of signal transportation, the execution of $ps_{10}$ requires that $ps_3$ and $ps_6$ have already been executed (since afterwards the amplifier has both required video signals that are transported to the television).

From the solution criteria of hybrid planning we can be sure that any plan step linearization respecting the ordering constraints and causal structure is an executable solution. However, some linearizations might be confusing to some
Finding User-friendly Linearizations of Partially Ordered Plans

users. Consider the plan in Fig. 4: The sequence \( ps_1, ps_5, ps_3, ps_6, ps_2 \) is a prefix of a valid linearization of its plan steps. However, the user repeatedly switches back and forth between different cables and devices in that linearization:

1. \( ps_1 \): plug DVI cable into blu-ray player
2. \( ps_5 \): plug scart end of scart-to-cinch cable into satellite receiver
3. \( ps_3 \): plug cinch cable into blu-ray player
4. \( ps_6 \): plug cinch-video end of scart-to-cinch cable into amplifier
5. \( ps_2 \): plug the other end of the DVI cable into amplifier

A more reasonable solution would be to execute \( ps_2 \) directly after \( ps_1 \) and \( ps_4 \) directly after \( ps_3 \), and so on. In the following, we introduce three different possibilities to choose reasonable execution orders. These are based on the actions’ parameters, on the causal link structure of the plan, and on the hierarchical structure of the planning domain.

4 Plan Linearization

This section gives three different utility functions for plan linearization. The intuition behind the following definition of the overall utility is that a good next action to execute has to fit into the context, i.e. the sequence of actions that have already been executed. Therefore the overall utility is a sum of local utilities between each action and its predecessors in the linearization.

**Definition 1 (Utility of Plan Linearizations)**

Given a plan \( P^* = (PS^*, <^*, VC^*, CL^*) \) and a linearization of its plan steps \( L = \langle ps_1, ps_2, \ldots, ps_n \rangle \) that is valid with respect to \(<^*\), we define the (overall) utility \( U(L) \) of the linearization as

\[
U(L) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} w(j) \cdot u(ps_i, ps_j^{-1})
\]

The additional weight function \( w(j) \) is introduced to allow for different factors, e.g. to give the recently executed actions more weight than those at the beginning of the execution. The following subsections give three possible definitions of that local utility function \( u(ps_i, ps_j) \). The first one is based on the actions’ parameters, the second on the causal links, and the last one on the domain’s task hierarchy.

4.1 Parameter Similarity

The first (local) utility function is entirely based on the steps in the plan. More precisely, it employs a pairwise comparison of two plan steps’ parameters. This is based on the assumption that the tasks represent things to do, while the parameters represent entities that are necessary to do so. In our example the user may want to complete connecting a specific cable or device before switching his attention to other devices or cables.
Definition 2 (Parameter-based Utility)
We define the parameter-based utility \( u^p \) of two ground plan steps \( ps^1(c_1^1, \ldots, c_{m^1}^1) \) and \( ps^2(c_1^2, \ldots, c_{n^2}^2) \) as
\[
u^p(ps^1(c_1^1, \ldots, c_{m^1}^1), ps^2(c_1^2, \ldots, c_{n^2}^2)) = \| \sum_{i=1}^{m^1} \sum_{j=1}^{n^2} u^c(c_i^1, c_j^2) \|
\]
where \( \| \| \) is an arbitrary normalization operator.

The given function compares each pair of the parameters of \( ps^1 \) and \( ps^2 \). A simple function to compare two parameters is given in Def. 3. More sophisticated functions could e.g. include the parameters’ sorts when they are not identical.

Definition 3 (Parameter Similarity)
\[
u^c(c_1, c_2) = \begin{cases} 1, & \text{if } c_1 = c_2 \\ 0, & \text{else} \end{cases}
\]

After summing up the different parameter similarities, the overall utility value of the plan steps has to be normalized to prevent the approach from preferring plan steps with many parameters. A straightforward realization of this function would consist in, for example, dividing the utility by the product of the parameter count of both plan steps (if greater than zero).

There are some self-evident extensions of the given utility definition, e.g. by including also the task schemata of the plan steps. In our domain, only a single task schema is used (there are only plugin actions), but in general several different task schemata are possible; depending on the domain it might be preferable to execute similar task schemata consecutively. For instance, in another domain, it might be plausible to execute all “shopping” actions together before (or after) executing “cleaning” actions.

4.2 Causal Link Structure
The next utility function is based on the plan’s causal link structure. Like POCL planning, our planning approach is problem-driven, i.e. every causal link in the plan is necessary to support some precondition. Thus the structure of the causal links represents the causal structure of the plan. This causality information can be utilized for plan linearization.

Definition 4 (Causal Link-based Utility)
We define the causal link-based utility of two plan steps \( ps^1 \) and \( ps^2 \) as
\[
u^l(ps^1, ps^2) = u^l(\{ \varphi \mid (ps^1 \rightarrow_{\varphi} ps^2) \in CL^* \})
\]
where \( u^l(\cdot) \) maps to an utility number.

The function \( u^l(\cdot) \) maps the set of causal links between two plan steps to a utility number. One possibility is the set cardinality. However, there are several alternatives, e.g. to return 1 if there is at least one link and 0, otherwise.

The given definition is based on the causal links between two plan steps. In combination with an appropriate weight function \( w \), this utility results in
linearizations where the steps are ordered in a way that the preconditions of some step are established (directly) preceding plan steps.

4.3 Decomposition Information

Since a planning domain is (usually) modeled by a human, the way the task hierarchy is modeled might also carry information. Partial plans that are used in decomposition methods are implementations of their abstract tasks [3]. Tasks that are within the same partial plan can hence be considered to be semantically related. We generalize this relationship between tasks in the plan of the same method to those of different methods and give another utility that is based on the Task Decomposition Graph (TDG). Our definition of the TDG is a simplified variant of the TDG that was used in earlier work [4].

Definition 5 (Task Decomposition Graph)
Let \( (V, E) \) be an AND/OR graph with the set of vertices \( V \) and the set of edges \( E \subseteq V \times 2^V \). We define the Task Decomposition Graph (TDG) as the minimal graph that fulfills the following specification:

1. base case:
   (a) \( t_{\text{init}} \in V \), where \( t_{\text{init}} \) is a new artificial root node
   (b) \( V \supseteq \bigcup_{T \in \text{Ground}(PS_{\text{init}}, VC_{\text{init}})} T \)
   (c) \( E \supseteq \bigcup_{T \in \text{Ground}(PS_{\text{init}}, VC_{\text{init}})} \{(t_{\text{init}}, T)\} \)

2. \( \forall \overline{t}(\overline{\tau}) \in V : (t(\overline{\tau}), \langle PS', \prec', VC', CL' \rangle) \in M \land \theta \) is mgu of \( \overline{\tau} \) and \( \overline{\tau}' \Rightarrow \)
   (a) \( V \supseteq \bigcup_{T \in \text{Ground}(PS', VC' \cup \theta)} T \) and
   (b) \( E \supseteq \bigcup_{T \in \text{Ground}(PS', VC' \cup \theta)} \{(t(\overline{\tau}), T)\} \)

Where \( \text{Ground}(PS, VC) \) denotes a set of sets. Each element is a set of ground tasks obtained by grounding the plan steps \( PS \) with respect to the variable constraints in \( VC \). The returned outer set contains all valid groundings.

The TDG includes nodes for every ground instance of every task in the initial task network (1b). To connect these nodes in the initial plan, a new node is introduced (1a) that is the parent node of them (1c). For every task in \( V \) and every compatible method, the groundings of the tasks in the method’s plan are also nodes in the graph (2a). And each compatible grounding of the method’s plan is connected via a multi-edge (a connector) to the parent task (2b).

Definition 6 (Decomposition-based Utility)
Let the decomposition-based utility between two plan steps \( l_1: t_1 \) and \( l_n: t_m \) be defined as

\[
h^d(l_1: t_1, l_n: t_m) = \min \{n + m | (t_1, \ldots, t_n), (t'_1, \ldots, t'_m), t'_1 = t_1, t'_m = t_m, \text{ and for } 1 < j \leq n \text{ holds } (t_{j-1}, V') \in E \text{ and } t_j \in V',
\]

\[
  \text{for } 1 < k \leq m \text{ holds } (t'_{k-1}, V') \in E \text{ and } t'_k \in V' \}
\]
The utility gives the minimal distance of two tasks in the possible decompositions from the initial plan. To reach one node from the other, one has to go up the tree until a node is reached that is a parent node of both of them. Afterwards one has to go down the tree until the second node is reached. Since here we return some kind of similarity, please be aware that this function has to be minimized (in contrast to the others). As this utility relies solely on decomposition it can, in the current form, not deal with task insertion (cf. solution criterion (1)).

When a planning system generated a solution, it has found one valid decomposition of the initial plan that resulted in the plan at hand. That decomposition, called Decomposition Tree (DT) [5], is quite related to our TDG. From a practical point of view the node distance can be taken from this tree. However, while the TDG gives a measure for the semantic similarity of tasks, the DT is affected by the preconditions and effects as well as by the planning process itself. It is just one valid decomposition, but might not be minimal.

5 Discussion

In this section we first discuss how the proposed utility functions would perform in the home theater domain. Afterwards, we discuss how the different linearizations could be evaluated in a field study with human users.

5.1 Linearization Behavior in the Home Theater Scenario

We will now discuss the proposed utility functions based on the example solution depicted in Fig. 1.

Parameter-based Utility. Recall that each action takes four arguments: the source hardware like a device (e.g., a blu-ray player) or a cable, its port (where also cables have “ports”, these are their ends), and the destination hardware and port. Let us assume the plan step \( ps_1 \) is picked first for being presented to the user. We regard it as plausible to continue with \( ps_2 \), because both plan steps are concerned with connecting the DVI cable. We know that both \( ps_1 \) and \( ps_2 \) have a common constant that represents this cable. We can assume that the parameter similarity-based utility function may select \( ps_2 \) after \( ps_1 \) rather than selecting \( ps_5 \), where completely different devices and cables (hence disjoint constants) are used.

Note that the question how ties are broken (in case two actions are similar with respect to their parameter similarity-based utility value) can also influence the resulting linearization. In our example, recall that \( ps_1 \) and \( ps_2 \) share exactly one constant: the one representing the DVI cable. However, \( ps_1 \) and \( ps_3 \) also share a constant: the one representing the blu-ray player. Thus, rather than selecting \( ps_2 \) after the execution of \( ps_1 \), also \( ps_3 \) might be selected. Both selection strategies might be regarded useful: the one completes connecting the cable that was used last, whereas the other completes connecting the device that was used last.
Finding User-friendly Linearizations of Partially Ordered Plans

Causal Link-based Utility. If one looks at the solution plan in Fig. 1 it seems quite natural that we execute \( ps_2 \) directly after \( ps_1 \) and, as another example, \( ps_6 \) to \( ps_8 \) directly after \( ps_5 \). Why is this the case? Because, as depicted in the figure, these actions are causally related to each other. In hybrid planning, these causal dependencies are modeled explicitly using causal links and can thus easily be used for plan linearization.

Again, assume \( ps_1 \) was picked first. Since the only plan step to which \( ps_1 \) is causally related is \( ps_2 \), that plan step is picked next. The next causally relevant plan step (for \( ps_2 \)) is \( ps_9 \). However, because \( ps_9 \) requires \( ps_6 \) to be executed first, another plan step must be chosen.

The question of tie-breaking is also quite important for this plan linearization strategy as can be seen with the next example. Assume \( ps_5 \) (putting the scart end of the scart-to-cinch-cable in the satellite receiver) is selected for execution after \( ps_2 \). The next candidates are \( ps_6 \) to \( ps_8 \) (which correspond to plugging the cinch ends of the scart-to-cinch-cable into the amplifier). Assuming \( ps_6 \) were selected next, we are free to select either \( ps_9 \) or \( ps_7 \) or \( ps_8 \) next. The first tie-breaking strategy corresponds to depth-first selection, whereas the latter correspond to breadth-first selection. Selecting in a depth-first manner corresponds to establishing the video signal for the television first. Selecting in a breadth-first manner would first complete connecting the scart-to-cinch cable before proceeding.

Decomposition-based Utility. The last linearization technique is based on the decomposition hierarchy of the planning domain. Suppose the planning problem is given by three initial abstract \( \text{connect} \) tasks each taking two devices, which need to be connected. For each abstract task, the decomposition methods contain pre-defined standard solutions for these tasks. For instance, the two plan step sequences \( \langle ps_1, ps_2 \rangle \) and \( \langle ps_3, ps_4 \rangle \) can be put together in a partial plan \( P \) that is used by a decomposition method \( m = \langle t(\bar{c}), P \rangle \) for the task \( t(\bar{c}) = \text{connect}(\text{bluray}, \text{amplifier}) \). Analogously, the sub plan in Fig. 1 involving the plan steps \( ps_5 \) to \( ps_8 \) can be defined as a standard solution to the task \( \text{connect}(\text{receiver}, \text{amplifier}) \).

Now, let us say \( ps_1 \) is picked first for execution. Since the decomposition distance to \( ps_2 \) and \( ps_3 \) is the same and smaller than the distance to \( ps_5 \), one of these two plan steps is picked first. So, all plan steps \( ps_1 \) to \( ps_4 \) will be presented for execution before any other plan step of the solution. For tie-breaking between these plan steps, further strategies need to be considered since all of them have the same decomposition distance according to Def. 6.

5.2 Empirical Evaluation

The ultimate goal of the presented techniques is to assist users in real-world scenarios. The suitability of the three different utility functions for plan linearization and their relative strengths and weaknesses therefore call for empirical investigation. Can one of them be considered preferable, or does this depend, for example, on the application domain? Such questions are shared with other
subfields in HCI, where several alternative forms of empirical evaluation have been employed (see e.g. [6,7]):

1. the output/behavior of a system is compared to human experts (requiring a metric to quantify the difference),
2. users provide a subjective rating for a system’s performance, or
3. users have to solve a task assisted by a system, and task performance is used as the test metric.

These three forms of empirical evaluation measure different objectives, akin to the ones presented in Sec. 1:

1. in how far a system succeeds in imitating humans,
2. a system’s performance as subjectively perceived by prospective users, or
3. an objective metric for a system’s capability in assisting users, for example, users’ performance in carrying out a plan presented using the techniques outlined in this paper, as measured, for example, in terms of time taken to carry out a plan.

For the evaluation of the three utility functions for plan linearization, a first step is to identify test cases where the different approaches lead to different solutions (i.e. plan linearizations). The most simple experimental design would follow the second approach to evaluation, namely to present these different plan linearizations to the participants of an experiment, whose task is to rate them for appropriateness.

More concretely, such a simple experiment might consist in randomly assigning participants to three groups, each of which is presented with different plan linearizations, all for the same sequence of plans. The use of the three utility functions for generating the linearizations is counterbalanced across the three groups, such that for each plan each of the three approaches is used for one third of the participants, allowing for the average ratings from the participants to be compared (i.e., a latin square type of experiment design). For the presentation of plan steps within such an experiment, several options are available, ranging from simple lists to speech output [2]. In order to avoid confounds, in an experimental setting the most simple one is clearly preferable (i.e., a bullet point list).

As to the task that the participants have to fulfill in the experiment, the most simple design would ask them to provide a subjective rating for the plan linearizations, similar to the approach in [1]. This would reveal only the perceived usefulness or plausibility (as judged by the participants), not an objective measure for the effect of a particular ordering of plan steps has on a user’s success in carrying out a suggested plan.

By contrast, an experimental design measuring participants’ performance in carrying out plans with steps ordered by different linearizations would offer greater practical validity, but might fail to deliver results for various reasons. For example, if the problem turns out to be too easy or too difficult in general, or if the way the generated plan steps are presented turns out to be unsuitable,
differences in the employed linearization might not show strongly enough to be measured reliably.

The remaining option for designing an experiment – the first item in the above list – would be to offer participants the possibility to arrange an initially unordered plan in the order they prefer to see, and to evaluate the employed linearization approaches according to their agreement with the most frequent solutions. However, one needs to be careful that the order produced by the participants is not unintentionally influenced by potential flaws in the experimental setup, for example if the method through which participants are introduced to the plan, its steps, and by which they are to manipulate and indicate the final order of steps is biased towards a particular solution.

Another problem to consider is that the success of the given approaches (first of all, the quality of the linearizations) depends on the way the domain is modeled and, in case of the causal link utility function, the planning process that is used to generate the plans. This is due to the fact that (some of) the decisions of the planning system are reflected in the causal links. Thus the relative performance of linearization metrics might change when they are applied in different domains. However, these domain-independent methods might be a good starting point for more elaborated domain-specific heuristics.

6 Conclusion

Though each linearization of a partially ordered solution plan solves the given planning problem, a system that interacts with a human user should come up with a linearization that is intuitive to her or him. We addressed this issue in the context of a system that helps the user to assemble a complex home theater. We introduced three utility functions that are based on different properties of the planning problem and its solution. The first one relies solely on the similarity of the steps in the given plan. The second and third function benefit from our hybrid planning approach: they exploit the causal link structure generated during the planning process, and the way in which a solution was decomposed from the initial abstract plan. Further we discussed how one would go on to empirically evaluate the proposed models.

Acknowledgment

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References


Introducing Hierarchy to Non-Hierarchical Planning Models – A Case Study for Behavioral Adversary Models

Louisa Pragst, Felix Richter, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo

Institute of Artificial Intelligence, Ulm University, Germany

Abstract. Hierarchical planning approaches are often pursued when it comes to a real-world application scenario, because they allow for incorporating additional expert knowledge into the domain. That knowledge can be used both for improving plan explanations and for reducing the explored search space. In case a non-hierarchical planning model is already available, for instance because a bottom-up modeling approach was used, one has to concern oneself with the question of how to introduce a hierarchy. This paper discusses the points to consider when adding a hierarchy to a non-hierarchical planning model using the example of the BAMS Cyber Security domain.

Keywords: Hybrid Planning, Hierarchical Planning, POCL Planning, Domain Modeling, Task Hierarchy, Abstractions

1 Introduction

This work is an extended abstract based on the Bachelor’s Thesis “Hybrid Planning in Cyber Security Applications” [5]. The thesis is mainly concerned with converting the classical planning domain Behavioral Adversary Modeling System (BAMS) by Boddy et al. [2] into a hybrid planning domain. Hybrid planning is a planning approach that fuses hierarchical planning with concepts from Partial Order Causal Link (POCL) planning. When introducing a hierarchy to a non-hierarchical planning domain, several issues arise. This work gives an overview about some techniques that may be applied to come up with such a hierarchy.

Foremost the non-hierarchical planning model BAMS is presented. Subsequently hybrid planning, the planning formalism into which BAMS is transformed, will be presented. The main part explicates what to consider when adding a hierarchy to a non-hierarchical planning model.

2 Behavioral Adversary Modeling System

The non-hierarchical domain model that was used as a basis for this work is called BAMS and was introduced by Boddy et al. [2]. Its purpose is to help detecting
possible attacks against a computer network, especially from malicious insiders. Generated solution plans for a BAMS planning problem are supposed to portray possible attacks against the given network. Those plans can then be analyzed and weak points of the computer network can be deducted.

The BAMS domain enables the modeling of a broad range of attacks. It includes physical attacks such as keylogging, but also malware attacks, network sniffing and even social exploits. The virtual and physical interconnection of computers can be modeled as well as email communication and encryption.

## 3 Hybrid Planning

Hybrid Planning \(^1\) is a planning formalism that fuses hierarchical planning with POCL Planning. Here, not only primitive tasks have pre- and postconditions, but also the abstract tasks. These conditions are conjunctions of literals. For primitive tasks, the conditions specify in which states they can be executed. The predicates used by the literals describe connections between objects (that are represented using \textit{constants}). For instance, in the BAMS domain, a specific human could be the administrator of a specific computer. Each constant is of a specific \textit{sort} that represents a class of constants, such as humans or computers. Sorts can be arranged in a hierarchy. Tasks describe the possible actions such as \textit{logging in} or \textit{writing an email}. Only primitive tasks may be executed directly. Abstract tasks are abstractions of one or more task sequences, so-called partial plans. For each such partial plan, the domain model contains a so-called decomposition method mapping the respective abstract task to that partial plan. An abstract task’s pre- and postconditions describe its intended meaning. Any partial plan of a task’s decomposition method needs to satisfy a legality criterion to ensure that it is an “implementation” of its abstract task. To allow more flexibility, the pre- and postconditions of abstract tasks may use \textit{abstract literals}, which are abstractions of ordinary literals. Those are defined by means of other (possibly abstract) literals using so-called \textit{decomposition axioms}.

The planning problem is given in terms of an initial partial plan \(P_{\text{init}}\) (possibly containing primitive and/or abstract tasks) that specifies an initial state as well as the goal properties that should hold after the execution of a solution. A solution is an executable plan \(P\) that satisfies the goal properties and that is a refinement of \(P_{\text{init}}\). Refinement means that \(P\) is obtained from \(P_{\text{init}}\) via decomposing abstract tasks (replacing them by their implementations) and the insertion of ordering and variable constraints, causal links, and, if desired, the insertion of tasks.

## 4 Introducing a Hierarchy

This section describes the most important points one should consider when adding a hierarchy to a non-hierarchical domain model. It can be used either for bottom up approaches of building a hierarchical model or in cases where
there already exists a non-hierarchical model that is to be transformed into a hierarchical (or hybrid) model.

Before we explain the deployed techniques, we want to mention that there is only very little work in the literature that is concerned with the topic of automatically inferring a hierarchy of tasks. The constructive proof of Thm. 5 by Erol et al. [3] shows how a classical planning problem can be translated into an HTN planning problem with the same set of solutions than the original one. Such a hierarchical domain does, however, not constitute a “meaningful” hierarchical model as it does not calculate abstractions of tasks, but merely simulates task insertion via decomposition. The paper “Automatically Generating Abstractions” [4] is dedicated to the automated generation of abstractions. However, the produced domain model is tailored to a given problem instance, while we aim at introducing a hierarchy that is problem independent. Furthermore, the approach aims at reducing the search effort for planners, while we focus on developing a domain than can easily be read and understood by humans.

4.1 Task Hierarchy
When building a hybrid planning domain based on a non-hierarchical model, one mainly focuses on abstracting tasks. Given below are two types of abstraction: merging alternatives and abstracting a task sequence.

Abstracting Alternatives. One possible kind of abstraction is to merge alternatives. In the simplest case, there are two alternatives, each being a single task. As an example, consider a task for logging in with a password and another one for logging in with an installed certificate. Both tasks are used for logging in, but they have slightly different preconditions. They can hence be merged into a single abstract task login. The two primitive tasks still remain in the domain model, but the additional abstract task login is introduced together with two decomposition methods, each for one of the alternatives. The pre- and postconditions of abstract tasks must reflect the conditions of its primitive sub plans with respect to the legality criterion of decomposition methods [3]. For that purpose, abstract literals may be introduced. We will give an example later.

Abstracting Task Sequences. Another possibility is the abstraction of a sequence of tasks that is often done in a specific way, like checking emails. The pattern is always the same: logging in, reading all emails, opening attachments and logging out. One can introduce an abstract task checkEmails that is a representative for this task sequence.

4.2 Sort Hierarchy
A hierarchy of sorts is not only useful for imposing a logical structure on a domain model, properly used, it can also help to reduce the search space. We do not go into details how to find a plausible hierarchy of sorts, in particular, because the concept is not specific to hierarchical planning approaches. For instance, PDDL also allows to define a hierarchy on types, which is PDDL’s equivalent of sorts.
4.3 Relation Hierarchy

Hybrid Planning uses abstract relations to model accurate pre- and postconditions of abstract tasks. Recall that any partial plan $P$ that is used by a decomposition method for $t$ must be an “implementation” of $t$ [1]. The implementation criterion is defined by means of the pre- and postconditions of the task $t$.

Consider the two primitive tasks mentioned earlier: one performs a login using a password and another using an installed certificate. The two tasks only differ in their preconditions: the former requires a password and the latter requires an installed certificate. The previously introduced abstract task login cannot use any of these preconditions, since they do not hold for both alternatives. Instead, we can introduce an abstract relation loginIsPossible that is used as precondition of the abstract task. Then, we add a so-called decomposition axiom that defines loginIsPossible as satisfied if and only if either of the two primitive relations is satisfied.

5 Summary

To build a hierarchy for a non-hierarchical planning model one has to consider the hierarchy of sorts, relations, and, most importantly, tasks. The hierarchy of sorts can be easily used to reduce the search space. The hierarchy of relations is needed to keep as much information as possible about pre- and postconditions of tasks at higher levels of the hierarchy. These abstract tasks can mainly be introduced by abstracting from different alternatives or by abstracting from sequences of tasks that are often performed together.

Acknowledgment

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Design and Redesign of University Course Programs

Ulrich John
HWTK – University of Applied Sciences,
Friedrichstr. 198, 10117 Berlin, Germany
ulrich.john@hwtk.de/ john@sir-john.de

Abstract: In this workshop paper, we outline some ideas and aspects of a project in the context of configuration and planning that we plan to start as a student project at our quite young university. The architecture and development of proper software for the design and redesign of university course programs is planned to be the core of the “whole project”. Based on that core, further software components can be developed later on in subsequent student projects where the resulting software system/ software landscape – together with properly redesigned business processes – should support significantly the efficiency, the attractiveness and the agility of our university. In our opinion, a great part of the expected project results/ project findings will be suitable for using in similar projects focusing on other company types to enable agility, transparency and efficiency of the business processes. The intension of the paper is to encourage discussions and to produce new findings regarding the possibilities of configuration and planning technologies in the context of business processes and programs.

1 Introduction

The composition of well-suited course programs – with global goals and constraints – and the continuous redesign/ optimization of them are very important and complex work contents in the organization of a university. This is especially the case if the university is a “young university” – like HWTK in Berlin - that is still in the growing-up phase.

Analyzing the process of design and redesign of course programs, we found out that the core problem is very similar to extended product-design and product-redesign tasks.

Starting in the nineties, we have developed the well-founded general constraint-based configuration model ConBaCon (cf. [Joh98, JoGe02]) and several model extensions (cf. e.g. [Joh00, JoGe04]) over the years. The extended model is suitable for automatic and interactive configuration and reconfiguration of structured objects, enables the proper fulfillment of hard and weak constraints (also of prioritized constraint hierarchies), allows the consideration of time constraints - like availability and delivery constraints - and supports the design of integrated subcomponents which should be developed as part
of the desired object. In addition, it is also possible to process preference rules and process knowledge on the one hand. On the other hand, optimization goals can be taken into account. The specification of the particular configuration problem can be generated either using a graphical editor or by using the declarative specification language ConBaCon-L (cf., e.g. [Ges04]). We have used our ConBaCon model respectively the model library for conceptual designs and for the realizing of several configuration software systems in the fields of technical products, manufacturing plants, supply nets, and business processes. In some cases the approach was combined with proper simulation components to investigate and ensure the correctness of the computed configurations.

Based on that knowledge, it is a self-evident idea to use our models and approaches also for realizing the desired course program design system.

If it is possible to use the software for the design of new study courses and for the redesign/ actualization of existing course programs over time, several extensions are reasonable that support the agility, transparency and efficiency of our university.

In the next section, we focus on the design and redesign of university course programs and on their significance for organizing and building up a university. In section 3, we outline some core information about the ConBaCon model which we have developed over the last years especially for using in the field of product configuration and product reconfiguration. Basing on that, in section 4, we sketch ideas for using ConBaCon also for the design and redesign of course programs. In section 5, some first ideas for extensions are given which are followed by the conclusion.

2 Design and Redesign of Course Programs

The design and cyclical actualization of university course programs are very important process tasks inside a university that were done normally in a collaborative manual process using “normal office software” like Excel and Word. The process itself is quite time-consuming and error-prone.

The content of a course program should reflect the state of the art, should be attractive for students and should be realistic regarding the possibilities of teaching and learning.

In addition, the course program is very essential for a successful accreditation of a study course.

2.1 Problem Aspects/ Problem Structure

Analysing the problem of design and redesign of university course programs, we can identify a lot of characteristics and constraints. A few of them are:

- Each University usually offers several study courses,
A study course could have several specialisations,

Each study course has its own course program which can/should be improved/redesigned over time,

Each study course must reach a given number of ECTS points,

Each course program consists of several courses; during the design process, we have to determine which courses should belong to the course program -usually, there will be some alternatives,

For each course, the number of lectures units is fixed,

Each course has finally a number of topics/subtopics which are a significant selection from the set of the possible course subjects; usually, there is sometimes some degree of freedom for the lectures in shaping the topic/subtopic teaching,

Topics/subtopics can be part of different courses; in this case, a special focus should be laid on the fitness regarding content and sequence,

Topics can also belong the several study courses, e.g. a subject could be a topic in a bachelor course program and in a master course program; if it is a consecutive master study course, the content regarding the topic must be more advanced or has to have other views than in the bachelor course,

For each course, the students have to pass examinations; in principal, there are several types of examinations; the type has to be fixed in the course program,

…

2.2 Solution Aspects

If we want to support the design process by software, we need to have/to produce a specification which contains all information regarding the above mentioned information categories and their dependencies/constraints. Based on this specification, the software should answer the following questions - among others - in interaction with the user/users:

Which courses should be contained in which course program? The given number of ECTS points must be reached.

Which topics and subtopics should be integrated in which courses? The given number of lecture time units of each course must be reached. The recommended self-study content must correspond to the lecture contents and should take the students the defined number of hours (correlating to the ECTS points). If possible, all relevant topics of the course subject should be integrated into the course.
Furthermore, it would be reasonable if the main research of the university should be taken into account with a special focus in the course.

- Which sequence should be chosen for the topics? Also the sequence of related topics in different courses of the same study course is relevant.
- How to balance the use of examinations types, whereat the students have reduced examination stress and a good learning effect?
- Which literature should be recommended for which course?

2.3 Special Aspects

At our private university HWTK, we have some special aspects which are also relevant for architecture of the desired software application:

- Usually, we have parallel study groups whereat the same course can be given in the same semester by different lectures for different study groups. Of course, we have than some individual differences originated by the lectures.
- For students of the dual study model, it is compulsory to attend a given percentage of all course lectures. Therefore, the attendance of the students has to be documented.
- The actual content of each lecture has to be documented.
- At HWTK, we have 3 types of study courses: full time, dual study and distance learning. It should be ensured, that the students are able to change the study types at the end of each semester.
- It is planned that students can study for one or two semesters at one of our partner universities. To support this, it would be nice to have some harmonization of the course programs at all partner universities. Because this goal tends to be not fully realistic, the differences should be determined to support the individual learning which should enable a good integration.

3 Product Configuration/ Design and Reconfiguration/ Redesign

Starting in the nineties, we have developed the general constraint-based configuration model ConBaCon (cf. [Joh98, JoGe02]) and several model extensions (cf., e.g. [Joh00, JoGe04]). The original development target was to develop a framework for powerful automatic and interaction-driven product configuration software systems.

In essence, specifications/ ontologies of valid technical products and their features will be taken as input of the configuration software (ConBaCon system, see Fig. 1). Main elements of those specifications are object/ module specifications whereat objects are
described with their attributes and their structure. Objects can consist of several components wherein some components can be optional. On the other hand, there may be specialization relations wherein several alternatives are specified for a component/object. Technical and logical relations, especially also regarding the existence of components in the solution product and regarding value parameters/features can be specified by constraints (cf. [Ges04]).

Based on the specifications and on maybe given user demands and preferences concerning the desired product, an internal constraint-based model will be generated by ConBaCon. This model represents the particular configuration/reconfiguration problem and is used for visualization, user interaction processing and solution calculation/optimization. The “model heart” are the automatically generated so-called consistency-ensuring constraints (cf. [Joh98]) that ensure a strong search space reduction. Each solution of the underlying constraint net represents a valid product configuration resp. its specification.

Some extensions of our approach allow the integration of design aspects into the pure configuration/reconfiguration tasks. Another extension is the possibility to process specified constraint hierarchies (in essence: weak constraints with priorities). Using them also allows to support reconfiguration processes.

Special process knowledge can be specified by additional rules that control the configuration process where the rules are transformed to constraints or are processed by rule execution components.
4 ConBaCon for Course Programs

Beside the above mentioned extensions for design aspects, we have also developed some ConBaCon-model extensions for integrating time constraints (cf. [JohGe02]) and for the design of supply nets and production nets.

In this section, we initially brainstorm on how to use the ConBaCon approach for the design and redesign of course programs.

4.1 Model Extensions

Similar to products, course programs have also a hierarchical structure. Course programs consist of courses. Courses belong to one study semester and consist of topics, subtopics, lecture units, self-study units and so on. Like in product specifications, we can also identify structure units which are optional, e.g., optional course topics/ subtopics whereat topics/ subtopics could be related by their content and context. In addition to pure product configuration, we have precedence constraints and elements – like topic units - that have durations. Regarding the durations, we have additional constraints, e.g., the sum of all topic-unit durations of a course must be equal (with some allowed tolerance) to the sum of all course lectures.

Because we have also constraints among different courses (and maybe also among courses of different course programs), we have to make some further model extensions which are similar to extensions for multi-product configuration.

After our first analyse phases, we think that all additional problem specifics of course program design – compared with product design – could be realized by additional model elements, especially by additional constraint variables, additional constraints and maybe by additional rule types that represent the special aspects of courses/ course programs.

4.2 Support of User Interactions

Already in the case of product configuration and product reconfiguration, the model-based support of high-flexible user interactions is essential. For the design and redesign of course programs this is even much more important. For instance, the user/ users have to make decisions regarding the occurrence of topics/ subtopics in courses/ course programs, regarding assignments of topics to courses, regarding the granularity of topic units, regarding the duration of topic units and self-study units and so on. In each case it must be possible to take back some of the given decisions. That way, the user/ users are able to do what-if analyses that should lead to good or even optimized solutions. Besides the system functionality to allow interaction “at mostly each parameter of the configuration/planning problem” it is best practise to provide different problem views for visualization and interaction support. In our domain, one view could be for instance the subtopic view, another view could be a lecture-oriented view of a selected course/ course program or a study-group oriented view.
For planning and configuration software, it is generally very important and advisable to provide functionalities for high-flexible powerful user-interaction. Some interesting aspects regarding this topic can be found for instance in [PrRi14].

4.3 Knowledge Management & Knowledge Actualisation

As basis for the course program design, we need a specification of the whole problem instance which could be correctly processed by the design software. Building those specifications can be done by using a proper specification language (cf. [Ges04]) or by using a tailored graphical editor. In the case that more than one user is allowed to specify problem aspects, we will have to reason about proper authorisation concepts, about a documentation functionality and about versioning.

A very important aspect of university course programs is the attractiveness for potential students and companies that usually pay the study fees for their students studying in the dual study mode. Among some other attractiveness features, it is essential that all “current topics” and all “current future themes” will be taken into account with an adequate amount of study time and granularity. Therefore at each time all current topics must be included in the specification. This can be supported by cyclical manual completion/adaptation of the problem specification. More sophisticated would be – in addition - the automatic supervision of relevant web sites – maybe the course programs of competitive universities. In the case of content changes, alerts will be sent to the responsible user together with “difference notions” as proposals for completions. Those “difference notions” could be enriched automatically by additional gathered/ prepared information from the World Wide Web.

Of course, we will have different versions of course programs and problem specifications over the years. It would be reasonable to save/ represent all of them – together with documentations of changings.

Regarding the computed course programs, transparency supporting features should be provided by the software system. E. g., it must be possible to see and validate all relevant features of the course programs like the number of ECTS points, the workload profiles, the examination profiles and so on. This key performance indicators resp. the ranges of them could be also used as adaptable parameters controlling the interactive design process.

For supporting of semiautomatic redesign processes, it would be recommendable to realize prioritization possibilities for topics and subtopics. Together with information regarding the duration possibilities, the priorities contribute in controlling the redesign processes. The ability of efficient redesign of university course programs is an essential feature – among others - for enabling a high degree of university agility.
5 Some Ideas for Extensions

Although a powerful design software for university course programs could be a very useful tool also as standalone software, we have the vision to design and use the tool as a component of a larger software system/ software landscape that support efficiency, transparency and agility of our university processes. All next software components should support – together with the course-design software - cooperative planning/ configuration. A next could be for instance time tabling software (cf. [Gol06]) and software for strategic resource planning. Other possible extensions would be on the one hand documentation functionalities for documenting each given lecture. If the documentation could be formalized mostly, it could be also used for detailed replanning of the following course lectures and also as information pool for the colleagues with the goal to harmonize the course contents. On the other hand, the integration of Web 2.0 functionalities seems to be interesting. Using those functionalities, the students could directly give feedback to lecture units, ask questions and formulate desires/demand for the next lectures of the course.

6 Conclusion

The composition of well-suited course programs – with global goals and constraints – and the continuous redesign/ optimization of them are very important and complex work contents in the organization of a university. This is especially the case if the university is a “young university” – like HWTK in Berlin - that is still in the growing-up phase.

In the paper, we have compared product configuration/ reconfiguration with the design and redesign of university course programs. We have argued that the constraint-based configuration model ConBaCon could be extended to be able to support the design and redesign of university course programs. Some main aspects of the regarding model extensions and some needed new software functionalities where outlined. These ideas should be a starting basis for student projects which aim at the development of a constraint-based course-program-design software as a central part of a university software landscape that should support efficient processes, directed transparency and agility.

References


