User-Centered Planning

Pascal Bercher, Daniel Höller, Gregor Behnke, and Susanne Biundo

Abstract User-centered planning capabilities are core elements of *Companion*-Technology. They are used to implement the functional behavior of technical systems in a way that makes those systems *Companion*-able – able to serve users individually, to respect their actual requirements and needs, and to flexibly adapt to changes of the user's situation and environment. This book chapter presents various techniques we have developed and integrated to realize user-centered planning. They are based on a hybrid planning approach that combines key principles also humans rely on when making plans: stepwise refining complex tasks into executable courses of action and considering causal relationships between actions. Since the generated plans impose only a partial order on actions, they allow for a highly flexible execution order as well. Planning for *Companion*-Systems may serve different purposes, depending on the application for which the system is created. Sometimes, plans are just like control programs and executed automatically in order to elicit the desired system behavior; but sometimes they are made for humans. In the latter case, plans have to be adequately presented and the definite execution order of actions has to coincide with the user's requirements and expectations. Furthermore, the system should be able to smoothly cope with execution errors. To this end, the plan generation capabilities are complemented by mechanisms for plan presentation, execution monitoring, and plan repair.

1 Introduction

Companion-Systems are able to serve users individually, to respect their actual requirements and needs, to flexibly adapt to changes of the user's situation and environment, and to explain their own behavior (cf. Chap. 1, the survey on *Companion*-Technology [19], or the work of the collaborative research centre SFB/TRR 62 [21]).

Pascal Bercher · Daniel Höller · Gregor Behnke · Susanne Biundo
Ulm University, Institute for Artificial Intelligence, e-mail: forename.surname@uni-ulm.de

A core element when realizing such systems is user-centered planning. This chapter presents various techniques we have developed and integrated to realize user-centered planning [18]. They are based on a hybrid planning approach [20] that combines key principles also humans rely on when making plans by refining complex tasks stepwise into executable courses of action, assessing the various options for doing so, and considering causal relationships.

Planning for *Companion*-Systems may serve different purposes, depending on the application for which the system is created. Sometimes, plans are just like control programs and executed automatically in order to elicit the desired system behavior; but sometimes they are made for humans. In the latter case, plans have to be adequately presented to the user. Since the generated plans impose only a partial order on actions, they allow for a highly flexible execution order. A suitable total order must be selected for step-wise presentation: it should coincide with the user's requirements and expectations. We ensure this by a technique that allows finding *user-friendly* linearizations [33].

In particular when planning for humans, a plan execution component must monitor the current state of execution so that the system can detect failures, i.e., deviations from the expected execution outcome. In such a case, the hybrid plan repair mechanism finds a new plan that incorporates the execution error [17, 9].

Companion-Systems assist users to complete demanding tasks, so the user may not understand the steps that the system recommends to do. In particular after execution errors, question might arise due to the presentation of a new plan. To obtain transparency and to increase the user's trust in the system, it is essential that it is able to explain its behavior. Therefor, the purpose of any action within a plan may be automatically explained to the user in natural language [53].

These user-centered planning capabilities of plan generation, plan execution and linearization, plan repair, and plan explanation are essential capabilities to provide intelligent user-assistance in a variety of real-world applications [18]. As an example, we integrated all those techniques in a running system that assists a user in the task of setting up a complex home theater [34, 9, 15]. The respective system and, in particular, the integration of the user-centered planning capabilities with a knowledge base and components for user interaction is described in Chap. 24. Here, we focus on the underlying planning capabilities and explain them in detail. We use the planning domain of that application scenario as a running example.

The hybrid planning framework is explained in Sect. 2. Section 3 is devoted to plan execution. Plan execution consists of various key capabilities when planning for or with humans: the monitoring of the executed plans to trigger plan repair in case of arising execution errors (explained in Sect. 4), the linearization of plans to decide which plan step to execute next, and the actual execution of the next plan step, which includes the adequate presentation to the user. Section 5 introduces the plan explanation technique that allows to generate justifications for any plan step questioned by the user. Finally, Sect. 6 concludes the chapter.

2 Hybrid Planning Framework

Hybrid planning [36, 20] fuses Hierarchical Task Network (HTN) planning [27] with concepts known from Partial-Order Causal-Link (POCL) planning [41, 50].

The smooth integration of hierarchical problem solving (inherited from HTN planning) with causal reasoning (inherited from POCL planning) provides us with many capabilities that are beneficial when planning for or with humans:

HTN Planning. In HTN planning, problems are specified in terms of abstract activities one would like to have accomplished. To do so, they have to be refined step-wise into more specific courses of action that can be executed by the user. This provides us with certain benefits:

- First of all, a domain expert has more freedom in modeling a domain. Often, expert knowledge is structured in a hierarchical way. Hence, it is often known to the expert what actions need to be taken in which order to accomplish some high-level goal. Such knowledge can easily be modeled by introducing a hierarchy among the available actions. Many real-world application scenarios are hence modeled using hierarchical planning approaches such as hybrid planning or the SHOP approach [44, 39, 18]. Further, a domain modeler can be assisted in the task of creating a hierarchical domain model by techniques that automatically infer abstractions for hierarchical planning [7].
- That action hierarchy may then be exploited for generating and improving explanations [53]. When the user wants to know about the purpose of a presented action during execution, the hierarchy can be used to come up with a justification.
- The hierarchy defined on the actions may also be exploited to come up with plausible linearizations of plans [33]. The actions in the plans are presented to a user one-by-one. Some linearization might be more plausible to a user than others. So, presenting those actions close to each other that "belong to each other" with respect to the action hierarchy might achieve reasonable results.
- The way in which humans solve tasks is closely related to the way hierarchical problems are solved by a planning system. That makes it more natural to a user to be integrated into the planning process [5], as it resembles his or her idea of problem solving. The integration of the user into this decision making process is called *mixed initiative planning*. It is presented in Chap. 7.

POCL Planning. In POCL planning, problems are specified in terms of world properties that one would like to hold. The problem is solved via analyzing causal dependencies between actions to decide what action to take in order to fulfill a required goal. The way in which plans are found and how they are represented can be exploited in various ways:

- The causal dependencies between actions within a plan are explicitly represented using so-called causal links. Analogously and complementarily to the exploitation of the hierarchy, these causal relations can be analyzed and exploited to generate explanations about the purpose of any action within a plan [53].

- The causal structure of plans may be used to find plausible linearizations of the actions within a plan. For instance, given there are causal relationships between two actions, it seems more plausible to present them after each other before presenting another action that has no causal dependencies to either of them [33].
- Finally, the POCL planning approach also seems well-suited for a mixed initiative planning approach, since humans do not only plan in a hierarchical manner, but also via reasoning about which action to take in order to fulfill a requirement that has to hold later on.

Hybrid planning combines HTN planning with POCL planning; it hence features all before-mentioned user-centered planning capabilities.

2.1 Problem Formalization

A hybrid planning problem is given as a pair consisting of a domain model $\mathscr{D}$ and the problem instance $\mathscr{I}$. The domain describes the available actions required for planning. The problem instance specifies the actual problem to solve, i.e., the available world objects, the current initial state, the desired goal state properties, and an initial plan containing the abstract tasks that need to be refined.

More specifically, a domain is a triple $\mathscr{D} = \langle T_p, T_a, M \rangle$. T_p and T_a describe the *primitive* and *abstract tasks*, respectively. The primitive tasks are also referred to as *actions* – those can be executed directly by, and hence communicated to, the user. Actions are triples $\langle a(\bar{\tau}), pre(\bar{\tau}), eff(\bar{\tau}) \rangle$ consisting of a name a that is parametrized with variables $\bar{\tau}$, a parametrized precondition *pre* and the effects *eff*. As an example, Eq. (1) depicts the name and parameters of an action of the home assembly task (see Chap. 24) for plugging the audio end of a SCART cable into the audio port of an audio/video receiver. In the depicted action, the parameters are bound to constants, which represent the available objects – in the example domain those are the available hi-fi devices, cables, and their ports.

$$plugIn(\text{SCART-CABLE}, \text{AUDIO-PORT}, \text{AV-RECEIVER}, \text{AUDIO-PORT}) \tag{1}$$

The precondition describes the circumstances under which the action can be executed, while the effects describe the changes that an execution has on the respective world state. Formally, preconditions and effects are conjunctions of literals that are defined over the variables $\bar{\tau}$. For instance, the (negative) literal $\neg used(\text{SCART-CABLE}, \text{AUDIO-PORT})$ is part of the precondition of the action depicted in Eq. (1). It describes that the audio port of the SCART cable may only be plugged into a port if it is currently not in use. The effects of that action mark the port as blocked. Abstract tasks syntactically look like primitive ones, but they are regarded to be not directly executable by the user. Instead, they are abstractions of one or more primitive tasks. That is, for any abstract task t, a so-called *(decomposition) method* $m = \langle t, P \rangle$ relates that task to a plan P that "implements" t [20, 13]. The

set of all methods is given by M. The implementation (or legality) criteria ensure that t is a legal abstraction of the plan P, which can be verified by comparing the preconditions and effects of t with those of the tasks within P. One can also regard it the other way round: the implementation criteria ensure that only those plans may be used within a method that are actual implementations for the respective abstract task. That way, a human user (in that case the domain modeler) can be actively supported in the domain modeling process – independently of whether he or she uses a top-down or bottom-up modeling approach.

Plans are generalizations of action sequences in that they are only partially ordered. They are knowledge-rich structures, because causality is explicitly represented using so-called *causal links*. Formally, a plan P is a tuple $\langle PS, V, \prec, CL \rangle$ consisting of the following elements: the set PS is referred to as *plan steps*. Plan steps are uniquely labeled tasks. Thus, each plan step $ps \in PS$ is a tuple $l : t$, with l being a label symbol unique within P and t being a task taken from $T_p \cup T_a$. Unique labeling is required to differentiate identical tasks from each other that are all within the same plan. The set V contains the variable constraints that (non-)codesignate task parameters with each other or with constants. Codesignating a variable with a constant means to assign the respective constant to that variable. Codesignating two variables means that they have to be assigned to the same constant. Non-codesignating works analogously. The set $\prec$ is a strict partial order defined over $PS \times PS$. The causal links CL represent causal dependencies between tasks: each link $cl \in CL$ is a triple $\langle ps, \varphi, ps' \rangle$ representing that the literal φ is "produced" by (the task referenced by) ps and "consumed" by ps'. Due to that causal link, the precondition literal φ of ps' is called *protected*, since the solution criteria ensure that no other task is allowed to invalidate that precondition anymore (see Solution Criterion 2c given below).

A problem instance $\mathscr{I}$ is a tuple $\langle C, s_{init}, P_{init}, g \rangle$ consisting of the following elements: the set C contains all available constants. The conjunction s_{init} of ground positive literals describes the initial state. We assume the so-called *closed world assumption*. That is, exactly the literals in s_{init} are assumed to hold in the initial state, while all others are regarded false. The conjunction of (positive and negative) literals g describes the goal condition. All these goal state properties *must* hold after the execution of a solution plan. Hence, g implicitly represents a set of world states that satisfy g. In particular when planning for humans, not all goals are necessarily mandatory. Instead, some of them might only be *preferred* by the user. That is, while some goals might be declared as non-optional (those specified by g), a user might also want so specify so-called soft goals that he or she would like to see satisfied, but that are regarded optional. A planner would then try to achieve those goals to increase plan quality, but in case a soft-goal cannot be satisfied, the planning process does not fail altogether. Some work has been done in incorporating such soft-goals into hierarchical planning in general [39, 55] and in hybrid planning in particular [8]. For the sake of simplicity, we focus on the non-optional goals in this book chapter. Note that this is not a restriction, since any planning problem with soft goals can be translated into an equivalent problem without soft goals [23, 37]. The initial plan P_{init} complements the desired goal state properties by the tasks that the user would like to have achieved. This plan may contain primitive tasks, abstract

tasks, or both. In addition, it contains two special actions a_{init} and a_{goal} that encode the initial state and goal description, respectively. The respective encoding is done as usual in POCL planning: a_{init} is always the very first action in every refinement of P_{init}, while a_{goal} is always the very last. The action a_{init} has no precondition and uses s_{init} as effect[1], while a_{goal} uses g as precondition and has no effect.

Solution Criteria. Informally, a solution is any plan that is executable in the initial state and satisfies the planning goals and tasks, i.e., after the execution of a solution plan P_{sol}, g holds, and P_{sol} is a refinement of P_{init} thereby ensuring that the abstract activities the user should accomplish (specified in P_{init}) have actually been achieved. More formally, a plan P_{sol} is a solution if and only if two criteria hold:

1. P_{sol} is a refinement of P_{init}. That is, one must be able to obtain P_{sol} from P_{init} by means of the application of the following refinement operators:
   a. ***Decomposition.*** Given a plan $P = \langle PS, V, \prec, CL \rangle$ with an abstract plan step $l : t \in PS$, the decomposition of the abstract task t using a decomposition method $m = \langle t, P' \rangle$ results in a new plan P'', in which $l : t$ is removed and replaced by P'. Ordering and variable constraints, as well as causal links pointing to or from $l : t$, are inherited by the tasks within P' [13]. This is a generalization of Def. 3 by Geier and Bercher [29] for standard HTN planning without causal links. This decomposition criterion ensures that the abstract tasks specified in P_{init} are accomplished by any solution. That criterion is the reason why HTN or hybrid planning is undecidable in the general case [27, 29, 1, 13]. It also makes the verification of plans (i.e., answering "is the given plan a valid solution to the given problem?") hard (NP-complete) even under severe restrictions [6, 13].
   b. ***Task Insertion.*** In hybrid planning, both primitive and abstract tasks may be inserted into a plan. Note that it is optional whether this feature is allowed or not. Allowing or disallowing task insertion might influence both the complexity of solving the planning problem [29, 2] and of the solutions themselves [31, 32]. Allowing task insertion allows for more flexibility for the domain modeler, as it allows to define partial hierarchical models [36, 2]. That is, the domain modeler does not need to specify decomposition methods that ensure that any decomposition is an executable solution, as the planner might insert tasks to ensure executability. Thus, allowing task insertion moves some of the planning complexity from the modeling process (which is done by a user/domain expert) to the planning process (which is done automatically).
   c. ***Causal Link Insertion*** **and** ***Ordering Insertion.*** Given two plan steps ps and ps', within a plan, a causal link can be inserted from any literal in ps's effect to any (identical) literal in the precondition of ps'. The parameters of the two literals become pairwise codesignated. Also an ordering constraint may be

[1] More technically, it uses not just s_{init} as effect, but – because s_{init} consists only of positive literals due to the closed world assumption – also all negative ground literals that unify with any negative precondition that are not contradicting s_{init}. Otherwise, there might be a negative task precondition literal that could not be protected by a causal link rooting in the initial state.

inserted between ps and ps'. Both these refinement options are inherited from standard POCL planning. They are a means to ensure the executability of plans [41, 50].

2. P_{sol} is executable in the initial state s_{init} and, after execution of that plan, the goal condition g is satisfied. Since s_{init} and g are encoded within P_{init} by means of the two special actions a_{init} and a_{goal}, respectively, and because P_{sol} is a refinement of P_{init} due to Solution Criterion 1, both planning goals can be achieved by using standard POCL solution criteria. Thus, $P_{sol} = \langle PS_{sol}, V_{sol}, \prec_{sol}, CL_{sol} \rangle$ is executable in s_{init} and satisfies g if and only if:

   a. ***All tasks are primitive and ground.*** Only primitive actions are regarded executable. Grounding is required to ensure unique preconditions and effects.
   b. ***There are no open preconditions.*** That is, for each precondition literal φ of any plan step $ps \in PS_{sol}$ there is a causal link $\langle ps', \varphi, ps \rangle \in CL_{sol}$ with $ps' \in PS_{sol}$ thereby protecting φ.
   c. ***There are no causal threats.*** We need to ensure that the literals used by the causal links are actually "protected". This is the case if there are no so-called *causal threats*. Within a primitive ground plan $P = \langle PS, V, \prec, CL \rangle$, a plan step ps is threatening a causal link $\langle ps', \varphi, ps'' \rangle \in CL$ if and only if the set of ordering constraints allows ps to be ordered between ps' and ps'' (that is, $\prec \cup \{(ps', ps), (ps, ps'')\}$ is a strict partial order) and ps has an effect $\neg\varphi$.

2.2 Finding a Solution

Hierarchical planning problems may be solved in many different ways [4], hence various hierarchical planning systems and techniques exist, such as SHOP/SHOP2 [43], UMCP [26], or HD-POP [52, edition 1, p. 374–375], to name just a few.

We follow the approach of the HD-POP technique. The resulting planning system, PANDA [14, Alg. 1], performs heuristic search in the space of plans via refining the initial plan P_{init} until a primitive executable plan has been obtained. The algorithm basically mimics the allowed refinement options: it decomposes abstract tasks (thereby introducing new ones into the successor plan), inserts new tasks from the domain (if allowed; cf. Solution Criterion 1b), and inserts ordering constraints and causal links to ensure executability. Hierarchical planning is quite difficult. In the general case, it is undecidable, but even for some quite restricted special cases, it is still at least PSPACE-hard [27, 29, 1, 2, 13]. During search, that hardness corresponds to the choice of which task to insert and which decomposition method to pick when decomposing an abstract task. In the approach taken by PANDA, these questions are answered by heuristics: each candidate plan is estimated in terms of the number of required modifications to refine it into a solution, or by means of the number of actions that need to be inserted for the same purpose [14].

For standard POCL planning, i.e., in case the initial plan P_{init} does not contain abstract tasks, there are basically two different kinds of heuristics. The first kind

bases on delete-relaxation[2], as this reduces the complexity of deciding the plan existence problem from PSPACE to P or NP, depending on the presence of negative preconditions [22] and whether the actions in the domain and the given plan become delete-relaxed or just those in the domain [11]. The respective heuristics are the *Add Heuristic for POCL planning* [57], the *Relax Heuristic* [46], and a variant of the latter based on partial delete-relaxation, called *SampleFF* [11]. The second kind of heuristics is not just one single POCL heuristic, but a technique that allows to *directly* use heuristics known from state-based planning in the POCL planning setting [10]. The technique encodes a plan into a classical (i.e., non-hierarchical) planning problem, where the POCL plan is encoded within the domain.

The idea of delete-relaxation has also been transferred to hierarchical planning. Here, the complexity of the plan existence problem is reduced from undecidable to NP or P, depending on various relaxations [3]. There is not yet an implementation of that idea, however. Instead, we developed the so-called *task decomposition graph* that is a relaxed representation of how the abstract tasks may be decomposed [25, 24]. That graph may both be used for pruning infeasible plans from the search space (i.e., plans that cannot be refined into a solution) [25] and for designing well-informed heuristics for hierarchical and hybrid planning [24, 14].

3 Plan Execution

In most real-world application domains, the effect of actions is not fully deterministic, though there is often an outcome that can be regarded as the intended or standard effect. Since *Companion*-Systems flexibly adapt to any changes in the user's situation and environment, they must be able to detect and deal with unforeseen effects. The sub system that monitors the environment and detects state changes that conflict with the current plan is called *execution monitor* and described in Sect. 3.1. When a state deviation is detected that may cause the current plan to fail, the *plan repair* component is started. The plan repair mechanism is introduced later on in Sect. 4. Solution plans are not totally ordered: they include only ordering constraints that are necessary to guarantee executability. Thus it is likely that there is more than one linearization of the solution. The *plan linearization* component is responsible to decide which one is most appropriate to be presented to a user. This functionality is described in Sect. 3.2. What it means to execute a single plan step, and how it may be done, is described in Sect. 3.3.

[2] Delete-relaxation means to ignore negative literals in the effects and, optionally, in the preconditions of any action.

3.1 Monitoring

As given above, the monitoring compares changes that have been detected in the environment with the intended effect of a started action. When differences are detected, it must not necessarily be a problem for the execution of the current plan, so the monitoring has to decide whether repair (see Sect. 4) is initiated or not. The decision may be based on the set of *active* causal links. A causal link is active if and only if its producer has been executed while the consumer has not. When there is an active link on a literal that has changed, repair is started (see Fig. 1).

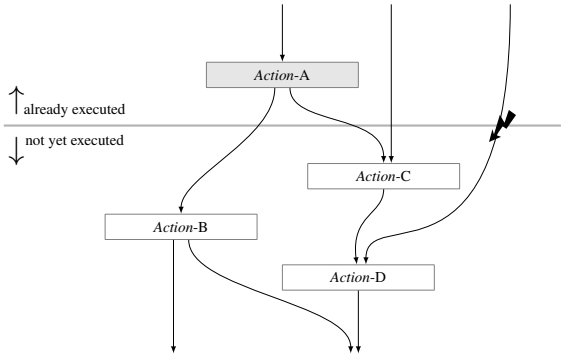


Fig. 1 The figure shows how an unexpected state deviation influences the execution of the remaining actions. The horizontal line indicates the execution horizon. An execution error flipped the truth value of the literal protected by the right-most causal link. Because that causal link crosses the execution horizon, the causal link's consumer (Action-D) might not be executable anymore.

Intuitively, this means that (a part of) the precondition of the consumer should have been fulfilled by the producer, but this has not been successful. Now there is no guarantee that the precondition of the consumer is fulfilled (i.e., a causal link that supports it) and plan execution may fail. There are special cases, however, where the currently executed sequence of actions is still executable although there is a causal link that is violated (a valid POCL plan could hence be found by simply choosing a new producer for the invalidated causal link within that plan). Although in that case the user could proceed executing that action sequence, plan repair must be initiated, since the respective causal link may be mandatory: in case it has not been inserted by the planner as a means to ensure executability, but if it comes from the domain (specified within a plan referenced by a decomposition method) or from the initial plan, it may not be changed. Such links may be intended by the modeler to protect certain properties during execution (referred to as prevail conditions) and are thus not allowed to be removed.

So, when ever a condition of an active causal link is violated, the plan monitoring initiates plan repair. It creates an altered plan (if there is one) that is able to deal with unexpected changes and fulfills the constraints given in the model.

The approach given above is able to deal with unforeseen changes of the environment and minimizes the computational effort that is necessary. Plan repair is only started if active causal links are violated. However, there are situations where it would be beneficial to start the repair mechanism even in cases where no active

causal links are violated and, hence, the plan is still executable. Consider, e.g., the case where the unforeseen change does not result in any violated causal link, but at the same time causes the original goal condition to become true. In that setting, the planning problem would be solved when no further actions are executed[3]. Without starting repair, the user had to proceed executing the plan. Though this is a good reason to start plan repair as often as there is enough time to wait for the new plan, there are also reasons to continue the execution of the original plan: in case there is no notable problem with the plan currently executed it might confuse the user when its execution is canceled to proceed another plan. The question when to repair could be answered by an empirical evaluation.

3.2 Plan Linearization

As given in the introduction of this section, plans generated by the planning system are only partially ordered. They include only the ordering constraints that are included in the model and those that have to be included to guarantee that a goal state is reached after execution. This makes the execution most flexible, since it commits only on necessary constraints. In many situations, it is necessary to choose a linearization of that partial order for plan execution. When plans are executed by a machine, like a smartphone or robot, it may not matter which of its linearizations is executed. However, whenever humans are involved in plan execution, the low commitment of the ordering given in the plan can be exploited to choose the linearization that is most suitable for the specific user in the current situation. Consider a user who has to achieve two tasks that are not related in any sense. This scenario is likely to result in a plan with two lines of action that are not interrelated. It is no problem to execute the first step of the first line, then the first step of the second line, and so on. However, it might be much more intuitive for the user to finish the first line before starting the second one (or vice versa). The overall process, committing on some ordering constraints during planning and determining the other ordering relations during post processing, can be seen as a model that consists of two parts.

There are several objectives for the linearization that may be competing. This could be the convenience of the user during execution, to optimize a metric that can be measured (e.g., execution time) or to imitate human behavior. Since *Companion*-Systems need to adapt to the specific user and its current situation, finding a user-friendly and maybe user- and situation-specific linearization is another point where adaptivity may come to light. As a starting point for situation- and user-specific strategies, we identified three domain-independent strategies to linearize plans [33].

All of them exploit knowledge that is included in the plan or the domain definition to linearize plans:

[3] Assuming there are no not yet executed actions that are inserted due to the underlying action hierarchy, cf. Solution Criterion 1a.

1. **Parameter Similarity.** In the home theater domain (see Chap. 24) it seems reasonable to complete all actions involving a specific device before starting on another. It is a feasible design decision of the modeler to pass on the devices as parameters to an action (though there are other ways to model the domain), as it is the case for the example action in Eq. (1). A parameter-based strategy would exploit this: it orders plan steps in a way that maximizes successive actions that share constants in their parameter set [33, Section 4.1].
2. **Causal Link Structure.** The causal link structure of a plan represents which effect of a plan step fulfills a certain precondition of another. The planning procedure is problem driven, i.e., there is no needless causal link in the plan. Therefore this is also a valuable source of linearization information, because the user may keep track of the causality behind steps that are executed. A strategy based on this structure orders the steps in a way that minimizes the distance between producer and consumer of a causal link. Besides the decisions of the domain modeler, this strategy also depends on the planning process [33, Section 4.2].
3. **Decomposition Structure.** Since the planning domain is commonly modeled by a human domain designer, it is reasonable to assume that tasks that are introduced by a single method are also semantically related. Generalizing this assumption, tasks that have a short distance in the tree of decompositions that spans from the initial task network to the actual plan steps are supposed to be semantically closer related than tasks that have a long distance. This property can be used for plan linearization. In this form, it depends on both the domain and the planning process. When using the task decomposition graph instead, it only depends on domain properties [33, Section 4.3].

As given above, all strategies depend on the planning domain, the planning system, or both. Thus it is possible to model the same application domain in such a way that they work well or poorly. Consider, e.g., the strategy based on parameter similarity in a propositional domain – there is no information included that could be used for linearization.

The given strategies can be used to pick the next plan step from a set of possible next actions (those where all predecessors in the ordering relation have already been finished), i.e., for a local optimization. Another possibility is to optimize them globally over the linearization of the whole plan. They can also be used as starting point for a domain-specific strategy.

3.3 Plan Step Execution

There are several possibilities on how to proceed when a single plan step has been selected for execution. In some cases, the action is just present due to technical reasons and nothing has to be done for its execution. Consider, e.g., the actions a_{init} and a_{goal}. Their purpose is to cause a certain change during the planning process and it is likely that they can be ignored by the execution system, although reaching action a_{goal} could trigger a notification that states the successful plan completion.

A second possibility is that actions control some part of the system. These are executed internally, but are not necessarily required to be communicated to the user. They may cause, for example, a light to be switched on/off, or a door to open/close, or adding a new entry to be added in a calendar.

Besides these possibilities, there are actions that have to be communicated to the user, as he or she is the one that carries them out or because the presentation itself is the desired purpose. Such actions can be easily communicated to the user by relying on additional system components taken from dialog management (Chap. 9) and user interaction (Chap. 10), as explained in Chap. 24. For that purpose, each action has an associated dialog model that specifies how it may be presented to a user [47, 16]. The dialog model may itself be structured in a hierarchical manner to enable the presentation of an action with a level of detail that is specific to the individual user. So, depending on the user's background knowledge, the action may be presented with more or less details [48]. The resulting information is sent to the fission component [35] that is responsible for selecting the adequate output modality (Chap. 10). For this, each action may have a standard text template associated with it, which can be used for visualization. Further, each constant used by an action parameter can be associated with respective graphics or videos. In Fig. 2 we see how the action given in Eq. (1) may be presented to a user.

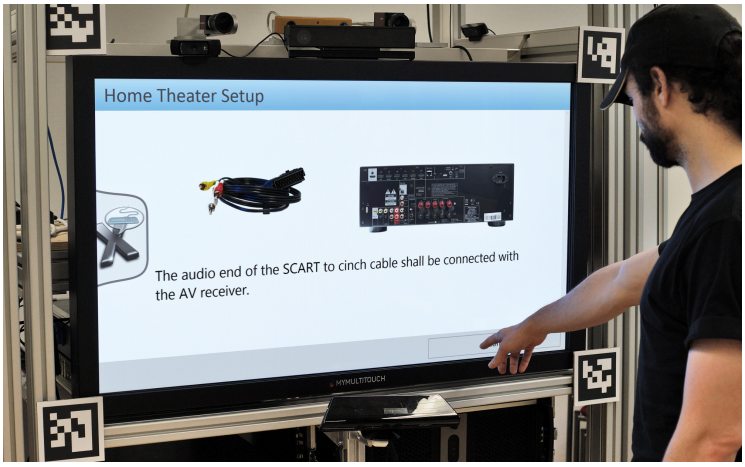


Fig. 2: Here, we see how a single planning action can be presented to a human user.

4 Repairing Failed Plans

Companion-Systems have to adapt to changes in the current situation [19]. This is especially necessary when the execution of a plan fails. If the plan monitoring

component (see Sect. 3.1) decides that – due to an execution failure – a new plan has to be found, there are two possibilities how this can be done:

- **Re-planning.** The plan at hand is discarded and the planning process is done from scratch. The changed environment is used as initial state and a new plan is found that transfers it into a state that fulfills the goal criteria.
- **Plan Repair.** The original plan is re-used and adapted to the needs of the changed situation. Thereby the unexpected changes of the environment have to be considered and to be integrated into the new plan.

Both approaches have several advantages and disadvantages. *Re-planning* enables the use of sophisticated planning heuristics. For some cases in classical planning, Nebel and Koehler showed that plan repair might be computationally more expensive than planning from scratch [45]. The system could come up with a completely new solution that has nothing in common with the original one, albeit a minor change would have resulted in a valid solution. Presenting a very different solution to a human user might cause confusion and reduce the user's trust in the system.

When a plan is *repaired*, the new plan might be more similar to the original solution. However, this strategy might increase computational complexity [45], prevents the planning system to find shorter/more cost-effective solutions; and an altered algorithm with effective heuristics that are able to deal with the altered planning problem has to be realized.

In HTN planning there is another aspect to consider: While in classical planning the already executed prefix of the original solution, followed by a completely new plan that reaches a goal state is a proper solution to the original problem, the combination may not be in the decomposition hierarchy of an HTN problem and thus violate Solution Criterion 1. There are circumstances that can be encoded into the decomposition hierarchy of an HTN planning problem that can not be ensured by preconditions and effects (see the expressivity analysis by Höller et al. [31, 32]). So it has to be assured that a repaired plan also fulfills the constraints that are introduced by the hierarchy.

We now introduce our approach for *re-planning* [9]. Although, from a theoretical point of view, it is classified as re-planning (because we do not try to repair the plan already found), it still combines aspects of both re-planning and plan repair. Aspects of repair are required to ensure that the plan prefix already executed is also part of any new solution that can cope with the execution error.

When the execution of a plan fails, a plan repair problem is created. Its domain definition is identical to that of the original problem, while the problem instance is adapted. It includes an additional set of obligations O, i.e., $\mathscr{I} = \langle C, s_{init}, P_{init}, O, g \rangle$. Obligations define which commitments that were made in the original plan have to be present in the new one. To ensure that they are fulfilled, we extend the solution criteria in such a way that all obligations need to be satisfied. There are obligations of the following kind:

- **Task Obligations.** These obligations ensure that a certain plan step (i.e., action) is present in the new solution. A task obligation is included into the problem for every step of the original plan that has already been executed. To overcome the

unexpected environment change, a special task obligation is added to the repair problem. It makes sure that a new action is added that realizes the unforeseen changes of the environment. Therefore it has the detected change as its effect. This action is called *process* [17] and is introduced after the executed prefix of the original plan.

- **Ordering Obligations.** Ordering obligations define ordering constraints between the obligated task steps.

Obligations from the given classes are combined in a way ensuring that the executed prefix of the original plan is also a prefix of any new plan. The process is placed exactly behind this prefix to realize the detected change of the world. So the new plan can cope with the unforeseen changes of the environment.

The additional constraints (i.e., the obligations) require some small alterations of the planning procedure. Given an unsatisfied obligation, the algorithm needs to provide possible refinements therefor: unsatisfied task obligations can be addressed via task insertion or decomposition and marking a task within a plan as one of those already executed. Ordering obligations are straight-forward.

We have also developed a *repair* approach for hybrid planning [17, 18]. It starts with the original planning problem and the set of refinements applied to find the original solution. As it is the case for our *re-planning* approach, the obligations are part of the planning problem as well to ensure that the execution error is reflected and the actions already executed are part of the repaired solution. In contrast to standard repair, the algorithm tries to re-apply all previously applied refinements. It only chooses different refinements where the particular choice leads to a part of the plan that cannot be executed anymore due to the execution failure.

5 Plan Explanation

Plans generated via automated planning are usually fairly complex and can contain a large amount of plan steps and causal links between them. If the decisions of a *Companion*-System are based upon such plans, its user may not immediately understand the behavior of the system completely. In the worst case, he or she might even reject the system's suggestions outright and stop using it altogether. In general, unexpected or non-understandable behavior of a cognitive system may have a negative impact on the trust in human-computer relationship [42], which in turn is known to have adverse effects on the interaction with the user [49]. To avert this problem, a system should be able to explain its decisions and internal behavior [40, 12]. If a planner is the central cognitive component of the system, it has to be able to explain its decisions (i.e., the plan it has produced) to the user.

A first step towards user-friendly interaction and eliminating questions of the users even before they come up is an intelligent plan linearization component (see Sect. 3.2), which presents the whole plan in an easy to grasp step-by-step fashion. Obviously, this capability is not sufficient for complete transparency. Although the order in which actions are presented is chosen in such a way that it is intuitive for

the user, he or she might still wonder about it or propose a rearrangement. The user might also be confused about the actual purpose of a presented action and ask why it is part of the solution in the first place. The hybrid plan explanation [53] technique is designed to convey such information to the user.

5.1 Generating Formal Plan Explanations

Usually, plan explanations are generated upon user request. Currently, the hybrid plan explanation technique supports two types of requests. The first inquires for the necessity of a plan step, i.e., "Why is action A in the plan?" or "Why should I do A?". The second requests information on an ordering of plan steps, i.e., "Why must action A be executed before B?" or "Why can't I do B after A?". In both cases the explanation is based upon a proof in an axiomatic system Σ, which encodes the plan, the way it was created, and general rules how facts about the plan can be justified [53]. The request of the user is transformed into a fact F and an automated reasoner is applied to compute a proof for $\Sigma \vdash F$. This proof is regarded as the actual formal explanation of the fact the user has inquired and is – subsequently – transformed into natural language by a dialogue management component and presented to the user [53, 9]. Obtaining such a proof in a *general* first order axiomatic system is undecidable. In our case it is decidable, since all necessary axioms are horn-formulas, i.e., disjunctions of literals with at most one being positive. This allows for the application of the well-known SLD-resolution [38] to find proofs.

We now describe which axioms are contained in Σ and how the inference, i.e., obtaining the formal proof, can be done. The plan itself is encoded in Σ by several axioms, using two ternary predicates cr and dr, which describe the causal and hierarchical relations, respectively. For every causal link $\langle ps, \varphi, ps' \rangle$ in the plan, the axiom $cr(ps, \varphi, ps')$ is added to Σ. As described in Sect. 2, the plan to be explained has been obtained by applying a sequence of modifications, i.e., by adding causal links, ordering constraints, tasks, or by decomposing abstract tasks. Each used method m was applied to decompose some abstract plan step ps'. It adds a set of new plan steps PS (and ordering constraints and causal links) to the plan. For every such plan step $ps \in PS$ the axiom $dr(ps, m, ps')$ is added to Σ.

Explaining the Necessity of Plan Steps. To answer the first kind of question, axioms proving the necessity of a plan step must be defined. That necessity is described using the unary predicate n. Note that by "necessity" we do not refer to an absolute or global necessity of a plan step. We do not answer the question whether the respective action has to be part of any solution (such actions are called *action landmarks* [51, 58]). Answering this question is in general as hard as planning itself. Instead, we explain the *purpose* of the action: we give a chain of arguments explaining for which purpose that action is used within the presented plan.

All plan steps of the initial plan P_{init} (which includes the action a_{goal} that encodes the goal condition) are necessary by definition, since P_{init} describes the problem it-

self. Thus, $n(ps)$ is included as an axiom for every plan step ps of P_{init}. If a plan step ps is contained in the plan in order to provide a causal link for another necessary plan step, ps is also regarded necessary, as it establishes a precondition of a required action. A simple example application of this rule is the necessity of any action establishing one (or more) of the goal conditions. The information that a plan step establishes the precondition of another plan step is explicitly given in hybrid plans by causal links. Using the given encoding of causal links in Σ, we can formulate an axiom to infer necessity as follows:

$$\forall ps, \varphi, ps' : cr(ps, \varphi, ps') \wedge n(ps') \rightarrow n(ps) \tag{2}$$

A similar argument can be applied if a plan step ps has been obtained via decomposition. If a necessary abstract plan step ps' is decomposed into ps, then ps serves the purpose of refining ps'. Converted into an axiom this reads:

$$\forall ps, m, ps' : dr(ps, m, ps') \wedge n(ps') \rightarrow n(ps) \tag{3}$$

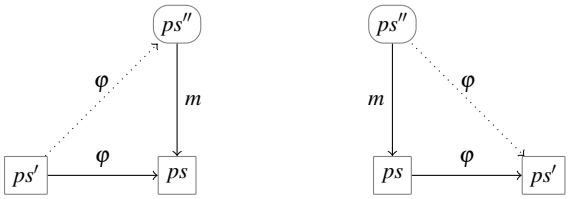
One can use both Axiom (2) and (3) to show the purpose of any plan step: it is either used to ensure the executability of another plan step (in this case, the first rule may be applied), or it is part of the plan because of decomposition (then, the second rule applies). Any chain of arguments (i.e., rule applications) will subsequently root in a plan step of the initial plan, i.e., the number of proof steps is always finite.

So far, the explanations based on causal dependencies (cf. Axiom (2)) do only rely on *primitive* plan steps. However, even these causality-based explanations could be improved when taking into account abstract tasks. E.g., the presence of the plan step *plugIn*(SCART-CABLE, AUDIO-PORT, AV-RECEIVER, AUDIO-PORT) should be explained as follows: it is necessary, as it is part of the abstract task *connect*(BLUERAY-PLAYER, AV-RECEIVER), which provides *signalAt*(AUDIO, AV-RECEIVER), which in turn is needed by the action *connect*(AV-RECEIVER, *TV*) to achieve the goal *signalAt*(AUDIO, *TV*). To obtain such explanations, *cr* predicates (i.e., causal links) involving abstract tasks must be inferred. Here, the idea is that if a plan step ps has an effect (or precondition) linked to some other plan step ps' that has been introduced into the plan by decomposing ps'', then ps'' is also linked to ps' as one of its primitive tasks generated the condition necessary for ps'. The axiomatic system Σ contains two further axioms, inferring these *cr* relations. Figure 3 contains a visual representation of both axioms.

$$\forall ps, m, ps'', ps' : dr(ps, m, ps'') \wedge cr(ps', \varphi, ps) \rightarrow cr(ps', \varphi, ps'') \tag{4}$$

$$\forall ps, m, ps'', ps' : dr(ps, m, ps'') \wedge cr(ps, \varphi, ps') \rightarrow cr(ps'', \varphi, ps') \tag{5}$$

Explaining the Order of Plan Steps. The second question a user might pose, i.e., why a plan step ps is arranged before some other plan step ps', has two possible answers. Either the order is contained in the plan presented to the user or it was chosen as part of the plan linearization process. In the latter case the system's answer could state that the order was chosen to obtain a plausible linearization and can be changed if the user wishes to. In the former case, again a proof for a fact is generated



(a) Visualization of Axiom (4) (b) Visualization of Axiom (5)

Fig. 3: The rectangular boxes depict primitive plan steps, the ones with rounded corners depict abstract plan steps. The arrows labeled with m indicate a performed decomposition using the method m. The arrows labeled with the literal φ indicate causal links, whereas the dotted ones are inferred by one of the Axioms (4) or (5).

and conveyed to the user. Necessary order between plan steps is encoded using the binary relation $<$. If the user poses the said question, the fact $ps < ps'$ is to be proven in Σ and its proof constitutes the formal explanation for the order's necessity. A necessary order can be caused by several reasons, each of which is described by an axiom. For the sake of brevity, we will only provide an intuition on these axioms, while the interested reader is referred to the work of Seegebarth et al. [53] for further details.

Orderings can be contained in the plans referenced by decomposition methods, thus they are necessary if the respective abstract task is. Further, an ordering constraint may be added to a plan if a causal threat is to be dissolved. Here, the necessity is based on the threatening plan step of the threat (cf. Solution Criterion 2c). Order is also implicitly implied by every causal link in the plan, and its necessity is based on the necessity of the consuming plan step of the link.

5.2 Verbalizing Plan Explanations

After having obtained a formal plan explanation, expressed by a proof in first-order logic, it has to be conveyed to the user in a suitable way. As a default-approach the explanation is transformed into text, which can be read to the user or displayed on a screen (see Fig. 2). To generate a natural language text, we use a pattern-based approach, an approach commonly used by automated theorem provers to present their proofs to humans [54, 30, 28]. Additionally one could use techniques similar to the Interactive Derivation Viewer [56], which uses both verbal and visual explanations.

Consider the example mentioned earlier in this section. The formal explanation in this case consists of one application of Axiom (3), two applications of the Axiom (2), one of Axiom (4), and two of Axiom (5). Resulting from this proof the following natural language text is generated:

> Plug the audio end of the SCART-to-Cinch cable into the AV Receiver to connect the Blu-ray Player with the AV Receiver. This provides that the AV Receiver has an audio signal,

needed to connect the AV Receiver with the TV. This provides that the TV has an audio signal, needed to achieve the goal.

We chose not to verbalize Axioms (4) and (5), as their application should be intuitively clear to the user. The remaining applications of Axiom (2) and (3) form a linear list. Each occurrence of Axiom (2) is translated into the text "This provides that $\langle \varphi \rangle$, needed to $\langle ps' \rangle$", where $\langle x \rangle$ denotes a domain-dependent verbalization of x. Likewise, each instance of Axiom (3) is translated into "Do this to $\langle ps' \rangle$" For the very first axiom in the explanation the begin of the sentences "This" and "Do this" are replaced with the verbalization of the action to be explained.

6 Conclusion

Flexible system behavior is essential when realizing *Companion*-Systems [19]. We summarized how different system capabilities supporting this design goal can be implemented using the hybrid planning approach, starting by the generation process that might integrate the user, the execution and communication of generated solutions, as well as discussing how to cope with unforeseen situations.

Though the current abilities of user-centered planning contributes valuable capabilities to the overall system, there are several promising lines of research for further improvements. Especially the problem of how plans are linearized [33] may offer further benefits for a convenient system. Another direction is a deeper explanation of system behavior [53, 9]. Here questions like "Why can't I use this action/method?" or an explanation on why a problem at hand has no solution may help the user. The overall explanation quality might also be improved by further integrating ontology- as well as plan-based explanations [7]. Another important matter in real-world applications is the presentation of different alternatives to reach a goal [12].

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

References

1. Alford, R., Bercher, P., Aha, D.: Tight bounds for HTN planning. In: Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 7–15. AAAI Press (2015)
2. Alford, R., Bercher, P., Aha, D.: Tight bounds for HTN planning with task insertion. In: Proc. of the 25th Int. Joint Conf. on AI (IJCAI), pp. 1502–1508. AAAI Press (2015)
3. Alford, R., Shivashankar, V., Kuter, U., Nau, D.: On the feasibility of planning graph style heuristics for htn planning. In: Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 2–10. AAAI Press (2014)
4. Alford, R., Shivashankar, V., Kuter, U., Nau, D.S.: HTN problem spaces: Structure, algorithms, termination. In: Proc. of the 5th Annual Symposium on Combinatorial Search (SoCS), pp. 2–9. AAAI Press (2012)

5. Behnke, G., Höller, D., Bercher, P., Biundo, S.: Change the plan - how hard can that be? In: Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 38–46. AAAI Press (2016)
6. Behnke, G., Höller, D., Biundo, S.: On the complexity of HTN plan verification and its implications for plan recognition. In: Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 25–33. AAAI Press (2015)
7. Behnke, G., Ponomaryov, D., Schiller, M., Bercher, P., Nothdurft, F., Glimm, B., Biundo, S.: Coherence across components in cognitive systems – one ontology to rule them all. In: Proc. of the 25th Int. Joint Conf. on AI (IJCAI), pp. 1442–1449. AAAI Press (2015)
8. Bercher, P., Biundo, S.: A heuristic for hybrid planning with preferences. In: Proc. of the 25th Int. Florida AI Research Society Conf. (FLAIRS), pp. 120–123. AAAI Press (2012)
9. Bercher, P., Biundo, S., Geier, T., Hörnle, T., Nothdurft, F., Richter, F., Schattenberg, B.: Plan, repair, execute, explain - how planning helps to assemble your home theater. In: Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 386–394. AAAI Press (2014)
10. Bercher, P., Geier, T., Biundo, S.: Using state-based planning heuristics for partial-order causal-link planning. In: Advances in AI, Proc. of the 36th German Conf. on AI (KI), pp. 1–12. Springer (2013)
11. Bercher, P., Geier, T., Richter, F., Biundo, S.: On delete relaxation in partial-order causal-link planning. In: Proc. of the 25th Int. Conf. on Tools with AI (ICTAI), pp. 674–681. IEEE Computer Society (2013)
12. Bercher, P., Höller, D.: Interview with David E. Smith. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0403-y
13. Bercher, P., Hller, D., Behnke, G., Biundo, S.: More than a name? on implications of preconditions and effects of compound htn planning tasks. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016), pp. 225–233. IOS Press (2016)
14. Bercher, P., Keen, S., Biundo, S.: Hybrid planning heuristics based on task decomposition graphs. In: Proc. of the 7th Annual Symposium on Combinatorial Search (SoCS), pp. 35–43. AAAI Press (2014)
15. Bercher, P., Richter, F., Hörnle, T., Geier, T., Höller, D., Behnke, G., Nothdurft, F., Honold, F., Minker, W., Weber, M., Biundo, S.: A planning-based assistance system for setting up a home theater. In: Proc. of the 29th Nat. Conf. on AI (AAAI), pp. 4264–4265. AAAI Press (2015)
16. Bertrand, G., Nothdurft, F., Honold, F., Schüssel, F.: CALIGRAPHI-creation of adaptive dialogues using a graphical interface. In: 35th Annual Computer Software and Applications Conf. (COMPSAC), pp. 393–400. IEEE (2011)
17. Bidot, J., Schattenberg, B., Biundo, S.: Plan repair in hybrid planning. In: Advances in AI, Proc. of the 31st German Conf. on AI (KI), pp. 169–176. Springer (2008)
18. Biundo, S., Bercher, P., Geier, T., Müller, F., Schattenberg, B.: Advanced user assistance based on AI planning. Cognitive Systems Research **12**(3-4), 219–236 (2011). Special Issue on Complex Cognition
19. Biundo, S., Höller, D., Schattenberg, B., Bercher, P.: Companion-technology: An overview. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0419-3
20. Biundo, S., Schattenberg, B.: From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In: Proc. of the 6th European Conf. on Planning (ECP), pp. 157–168. AAAI Press (2001)
21. Biundo, S., Wendemuth, A.: *Companion*-technology for cognitive technical systems. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0414-8
22. Bylander, T.: The computational complexity of propositional STRIPS planning. AI **94**(1-2), 165–204 (1994)
23. Edelkamp, S.: On the compilation of plan constraints and preferences. In: Proc. of the 16th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 374–377. AAAI Press (2006)
24. Elkawkagy, M., Bercher, P., Schattenberg, B., Biundo, S.: Improving hierarchical planning performance by the use of landmarks. In: Proc. of the 26th Nat. Conf. on AI (AAAI), pp. 1763–1769. AAAI Press (2012)

25. Elkawkagy, M., Schattenberg, B., Biundo, S.: Landmarks in hierarchical planning. In: Proc. of the 20th European Conf. on AI (ECAI), pp. 229–234. IOS Press (2010)
26. Erol, K., Hendler, J.A., Nau, D.S.: UMCP: A sound and complete procedure for hierarchical task-network planning. In: Proc. of the 2nd Int. Conf. on AI Planning Systems (AIPS), pp. 249–254. AAAI Press (1994)
27. Erol, K., Hendler, J.A., Nau, D.S.: Complexity results for HTN planning. Annals of Mathematics and AI **18**(1), 69–93 (1996)
28. Fiedler, A.: P.rex: An interactive proof explainer. In: Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR), pp. 416–420. Springer (2001)
29. Geier, T., Bercher, P.: On the decidability of HTN planning with task insertion. In: Proc. of the 22nd Int. Joint Conf. on AI (IJCAI), pp. 1955–1961. AAAI Press (2011)
30. Holland-Minkley, A.M., Barzilay, R., Constable, R.L.: Verbalization of high-level formal proofs. In: Proc. of the 16th Nat. Conf. on AI and the 11th Innovative Applications of AI Conf. (AAAI/IAAI), pp. 277–284. AAAI Press (1999)
31. Höller, D., Behnke, G., Bercher, P., Biundo, S.: Language classification of hierarchical planning problems. In: Proc. of the 21st European Conf. on AI (ECAI), pp. 447–452. IOS Press (2014)
32. Höller, D., Behnke, G., Bercher, P., Biundo, S.: Assessing the expressivity of planning formalisms through the comparison to formal languages. In: Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 158–165. AAAI Press (2016)
33. Höller, D., Bercher, P., Richter, F., Schiller, M., Geier, T., Biundo, S.: Finding user-friendly linearizations of partially ordered plans. In: 28th PuK Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK) (2014)
34. Honold, F., Bercher, P., Richter, F., Nothdurft, F., Geier, T., Barth, R., Hörnle, T., Schüssel, F., Reuter, S., Rau, M., Bertrand, G., Seegebarth, B., Kurzok, P., Schattenberg, B., Minker, W., Weber, M., Biundo, S.: Companion-technology: Towards user- and situation-adaptive functionality of technical systems. In: Int. Conf. on Intelligent Environments (IE), pp. 378–381. IEEE (2014). URL http://companion.informatik.uni-ulm.de/ie2014/companion-system.mp4
35. Honold, F., Schüssel, F., Weber, M.: Adaptive probabilistic fission for multimodal systems. In: Proc. of the 24th Australian Computer-Human Interaction Conf. (OzCHI), pp. 222–231. ACM (2012)
36. Kambhampati, S., Mali, A., Srivastava, B.: Hybrid planning for partially hierarchical domains. In: Proc. of the 15th Nat. Conf. on AI (AAAI), pp. 882–888. AAAI Press (1998)
37. Keyder, E., Geffner, H.: Soft goals can be compiled away. Journal of AI Research (JAIR) **36**, 547–556 (2009)
38. Kowalski, R.A.: Predicate logic as programming language. In: IFIP Congress, pp. 569–574 (1974)
39. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Proc. of the 5th European Semantic Web Conf. (ESWC), pp. 629–643. Springer (2008)
40. Lyons, J.B., Koltai, K.S., Ho, N.T., Johnson, W.B., Smith, D.E., Shively, R.J.: Engineering trust in complex automated systems. Ergonomics in Design **24**(1), 13–17 (2016). DOI 10.1177/1064804615611272
41. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proc. of the 9th Nat. Conf. on AI (AAAI), pp. 634–639. AAAI Press (1991)
42. Muir, B.M.: Trust in automation: Part I. theoretical issues in the study of trust and human intervention in automated systems. Ergonomics **37**(11), 1905–1922 (1994)
43. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of AI Research (JAIR) **20**, 379–404 (2003)
44. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Wu, D., Yaman, F., Muñoz-Avila, H., Murdock, J.W.: Applications of SHOP and SHOP2. Intelligent Systems, IEEE **20**, 34–41 (2005)
45. Nebel, B., Koehler, J.: Plan reuse versus plan generation: A theoretical and empirical analysis. Artificial Intelligence **76**(1-2), 427–454 (1995)
46. Nguyen, X., Kambhampati, S.: Reviving partial order planning. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI), pp. 459–466. Morgan Kaufmann (2001)

47. Nothdurft, F., Bertrand, G., Heinroth, T., Minker, W.: GEEDI - guards for emotional and explanatory dialogues. In: 6th Int. Conf. on Intelligent Environments (IE), pp. 90–95. IEEE (2010)
48. Nothdurft, F., Honold, F., Zablotskaya, K., Diab, A., Minker, W.: Application of verbal intelligence in dialog systems for multimodal interaction. In: 10th Int. Conf. on Intelligent Environments (IE), pp. 361–364. IEEE (2014)
49. Parasuraman, R., Riley, V.: Humans and automation: Use, misuse, disuse, abuse. Human Factors: The Journal of the Human Factors and Ergonomics Society **39**(2), 230–253 (1997)
50. Penberthy, J.S., Weld, D.S.: UCPOP: A sound, complete, partial order planner for ADL. In: Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR), pp. 103–114. Morgan Kaufmann (1992)
51. Porteous, J., Sebastia, L., Hoffmann, J.: On the extraction, ordering, and usage of landmarks in planning. In: Proc. of the 6th European Conf. on Planning (ECP), pp. 37–48. AAAI Press (2001)
52. Russell, S., Norvig, P.: Artificial Intelligence – A Modern Approach, 1 edn. Prentice-Hall (1994)
53. Seegebarth, B., Müller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In: Proc. of the 22nd Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 225–233. AAAI Press (2012)
54. Simons, M.: Proof presentation for isabelle. In: Proc. of the 10th Int. Conf. on Theorem Proving in Higher Order Logics (TPHOLs), pp. 259–274. Springer (1997)
55. Sohrabi, S., Baier, J.A., McIlraith, S.A.: HTN planning with preferences. In: Proc. of the 21st Int. Joint Conf. on AI (IJCAI), pp. 1790–1797. AAAI Press (2009)
56. Trac, S., Puzis, Y., Sutcliffe, G.: An interactive derivation viewer. Electronic Notes Theoretical Computer Science **174**(2), 109–123 (2007)
57. Younes, H.L.S., Simmons, R.G.: VHPOP: Versatile heuristic partial order planner. Journal of AI Research (JAIR) **20**, 405–430 (2003)
58. Zhu, L., Givan, R.: Heuristic planning via roadmap deduction. In: IPC-4 Booklet, pp. 64–66 (2004)