A Paradigm for Coupling Procedural and Conceptual Knowledge in Companion Systems

Marvin Schiller*, Gregor Behnke*, Mario Schmautz*, Pascal Bercher*, Matthias Kraus*, Michael Dorna†, Wolfgang Minker*, Birte Glimm*, and Susanne Biundo*

*Faculty of Engineering, Computer Science and Psychology, Ulm University, Ulm, Germany
†Corporate Research, Robert Bosch GmbH, Renningen, Germany

***Abstract*—Companion systems are technical systems that adjust their functionality to the needs and the situation of an individual user. Consequently, companion systems are strongly knowledge-based. We propose a modelling paradigm for integrating procedural and conceptual knowledge which is targeted at companion systems that require a combination of planning and reasoning capabilities. The presented methodology couples the hierarchical task network (HTN) planning formalism with an ontology-based knowledge representation, thereby minimising redundancies in modelling and enabling the use of state-of-the-art reasoning and planning tools on the shared knowledge model. The approach is applied within a prototype of a companion system that assists novice users in the do-it-yourself (DIY) domain with the planning and execution of home improvement projects involving the use of power tools.**

I. Introduction

Companion technology aims at making technical devices really smart. Such *companion systems* are situation- and user-adaptive assistants that provide their functionality in a completely individualised way to their users [1], [2]. For being able to provide their multitude of functionalities, these systems base, among others, on two fundamental and diametrical kinds of knowledge that need to be incorporated into the models of the application domain:

Firstly, the set of available tasks (which are either operations to be carried out by the system itself, or by the system's user to be presented to him or her as instructions), how they contribute to the user's goals, and what constraints exist among them has to be known. They are the basis for the system to plan ahead and to present available options to the user.

Secondly, the objects and concepts in the application domain, together with their semantic relationships, need to be represented in such a *companion system*.

Both the fields of automated planning and semantic knowledge representation have brought forth their own dedicated formalisms. Hence, when both are used in a *companion system*, redundancies, or worse, inconsistencies between the models of the application domain might arise. Here, we propose a methodology that serves to tightly integrate a hierarchical planning formalism with ontology-based knowledge representation. It is based on the idea that the static relations between objects (i.e. the planner's state) are represented in an ontology in such a way that the tasks of the planning model (describing the dynamics of the domain) directly incorporate the represented conceptualisations. Our paradigm strives to minimise any redundancies in modelling the planning domain and the modelling of conceptual knowledge. This facilitates coherence between the procedural and factual knowledge in the system's portfolio and helps to ease the issue of maintenance that arises when developing complex modularised systems. While the methodology itself is not limited to any particular application domain, we employ it in the context of a *companion system* which supports novice users with home improvement projects that require the use of electric power tools, such as electric drills, saws, and sanding machines. It represents a further development step w.r.t. our previous work in the domain of home appliances (e.g. setting up a home theatre [3], [4]) and the planning of fitness trainings [5].

The problem of combining planning with conceptual knowledge has previously been identified and different approaches have been proposed based on description logic. However, differently from our approach, some of these directly integrate description logics into the planning formalism, with the disadvantage that standard planning heuristics cannot be used. Some further approaches consider only a limited form of integration of ontology modelling and reasoning into planning. A short survey and comparison is presented in Sect. III.

II. Application Scenario

In our ongoing project, we have implemented a first demonstrator of a *companion system* which assists novice users in do-it-yourself (DIY) home improvement activities using power tools. The domain of home improvement provides an abundance of different activities that are carried out under varying circumstances (e.g. available tools, attachments and consumables) which can often be achieved by a variety of means. The assistance is given on the operation of individual power tools and the usage of the tools for different DIY activities. Furthermore, the assistant system supports a user while performing DIY projects, for which we use hierarchical planning to be able to instruct the user step-by-step at different levels of granularity. Further assistance is given on demand when a user asks for specific support. In the future, proactive support based user monitoring is planned, taking the individual situation and user's preferences into account.

As opposed to pure instructions, we focus on enabling the user to gain expertise in the home improvement domain

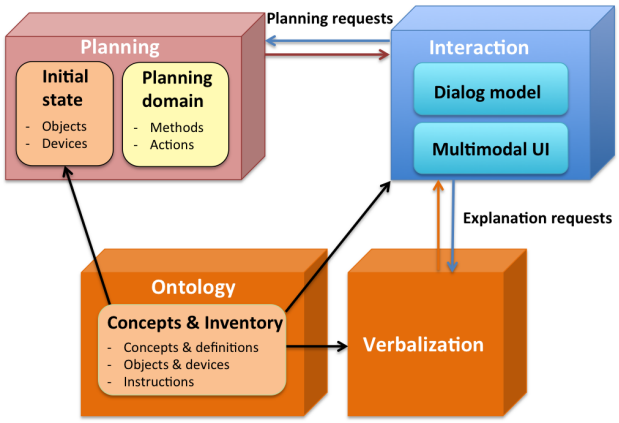

Fig. 1. Component overview

and to be productive in the future with the used tools. This includes the selection of tools which can be used for a given sequence of activities, the knowledge of suited attachments and accessories, also by taking properties of the used materials into account. To give a simple example, when connecting two work pieces with a screw, it may be necessary to pre-drill (e.g. for hardwood), to use a suitable screw for the material, etc. Again, drilling requires appropriate machine settings, an appropriate attachment (drill-bit) and a sufficiently charged battery for a cordless tool. The system has to provide suitable explanations dynamically, to offer useful information on the relevant concepts and properties, and to encourage the user to ask questions about the proposed steps. Hence a tight integration of task planning with conceptual knowledge representation is required, for which we are using an ontology and reasoning. An overview of the main components of the system architecture is shown in Fig. 1, where the coupling between ontology and planning represents the focus of the remainder of this paper.

III. Related Work

Gil [6] provides an extensive survey of research that connects description logics and planning. She groups the approaches into four categories: those reasoning about objects, actions, plans, and goals, respectively. The techniques we present fall into her categories of "reasoning about objects". This category contains systems where description logics is used to enhance the modelling of objects, their interconnections, and thus how states are described.

Our technique differs significantly from the previous ones. In the past, researchers aimed for a tight integration of descriptions logics into the planning process, which was achieved by using description logics as the formalism for state representation, instead of the purely propositional representation commonly used in planning. This greatly expands the expressiveness of the modelling language, but also incurs several disadvantages. First, specifying the semantics of actions becomes more complicated, as a delete-effect of an action cannot simply remove a fact f from the state, because that fact may have been inferred from others. Here, the delete effect would have to delete some *other* fact, such that f cannot be derived any more. Determining and selecting such a fact is a non-trivial problem, but was tackled e.g. by Ahmetaj et al. [7]. Second, changing the state representation makes almost all heuristics developed for planning useless, as they rely on a propositional state representation. We are only aware of the research by Sánchez-Ruiz et al. [8] on case-based planning with state models described in description logics, which might lead to pattern-database heuristics for such planning formalisms. However, modern planning is only efficient with such heuristics. Our techniques clearly separate the ontology and the planning model, while leveraging and combining their advantages.

More recently, other ways for combining description logics and planning have been proposed. Our previous work [5] falls into the line of action taxonomies, where we represent action hierarchies in a description logic framework and show how additional decomposition methods can be inferred automatically using reasoning. Sirin [9] propose a way to use hierarchical planning to find suitable decompositions of web services described in OWL-S, which is based on description logics. Hartanto and Hertzberg [10], [11] propose to use an ontological description of objects to prune those that are not actually necessary for solving a given planning task, thereby reducing the search space of the planner. Freitas et al. [12] consider how an HTN planning domain can be represented within an ontology formalism. This transformation allows one to use existing tools for ontologies to display the planning domain as formalised in description logics and to use reasoning tools on it. However, this work does not address how an interplay of procedural and factual knowledge within the application domain can be achieved. Thus, semantic information is not used to reason *within* the application domain and to integrate semantic information into the planning process. One recently developed framework that uses reasoning for connecting an ontology knowledge base to a planner is KMARF [13], where so-called "model transformation rules" are used to translate descriptions of states and planning operators to (non-hierarchical) PDDL problems to be solved by a planner. However, besides the architecture of the system, no details are offered about how these rules are defined and what modelling conventions need to be adhered to.

Our application scenario is shared with other approaches that propose assistance for users when interacting with technical devices, e.g. in the smart home domain (e.g. [14]). Planning for service composition in smart homes has been used in the SM4ALL project [15]. Georgievski et al. [16] explore a smart home system where an ontology is used for activity recognition and planning for controlling household devices such as lighting with the goal of saving energy. Here, information from activity recognition is used for planning, but a further integration between ontology and planning domain is not considered. On the other hand, Krieg-Brückner et al. [17] use ontology modelling to assist users to plan and prepare meals. Planning is performed by the SHIP-tool, which updates

states described in description logics, as opposed to our work that connects an ontology to a standard planning system using a propositional state representation. Their planner takes the role of a control program that selects the next action to be performed based on queries to the current ontology. However, this approach does not use modern planning heuristics, since these are not available for ontologies as state representations.

IV. Preliminaries

For reasoning about possible courses of action to reach the user's goals, we rely on a hierarchical planning formalism, which offers many advantages, in particular in the context of providing advanced user support based on planning technology [18].

The most fundamental ingredient to planning are tasks. By executing a task, the system or the user may transform one world state into another. In hierarchical planning, two kinds of tasks exist: primitive tasks and abstract tasks. A primitive task specifies a state transition, whereas an abstract task can be seen as a standard recipe for how to carry out more abstract activities [19]. I.e., an abstract task has to be refined into further courses of action.

More formally, a *primitive task* is a 3-tuple $\langle t(\bar{\tau}), pre(\bar{\tau}), \mathit{eff}(\bar{\tau})\rangle$ consisting of the task's name t and a sequence of task parameters $\bar{\tau}$. In our running example, we might have a task called $t = \mathit{AttachBattery}$ for describing how to attach a battery (cf. Fig. 4). The task parameters are variables, each associated with a sort; in our example, $\tau_1 = ?t$ is of sort *Tool* and $\tau_2 = ?b$ is of sort *Battery*. The planning domain contains a set of constants, which describe the available objects. Constants are grouped into (not necessarily disjoint) sorts. The task parameters are used by the preconditions $pre(\bar{\tau})$ and effects $\mathit{eff}(\bar{\tau})$ of the task. In its most simple form, preconditions and effects are simply conjunctions of literals over the task parameters. For instance, in the given example, the task's effect is simply the single literal (*AttachedBattery* $?t$ $?b$). For practical purposes, we allow quantifiers as well. E.g., a precondition of the task is (*not*(*exists*($?c$ − *Battery*)(*AttachedBattery* $?t$ $?c$))), quantifying over all constants of the sort *Battery*. We extended the planning domain description language PDDL [20] to not only allow the quantification over variables of a *specific* sort, but also over sorts themselves. E.g., a further precondition is (*exists*($?tt$ − *Type* [...])[...](*typeOf* $?t$ $?tt$)) stating that there needs to exist a sort $?tt$, such that the tool $?t$ is of that sort (note that "type" is the PDDL nomenclature for sort). We call a task *ground* if all its parameters are bound to constants. A ground primitive task is referred to as an action. Given the preconditions hold in a state, which is a conjunction of positive ground literals, the application of an action results in a successor state in which all negative effects are removed and all positive effects are added.

Whereas primitive tasks specify state transitions, *abstract tasks*[1] describe how to carry out more complex activities. For this purpose, the planning domain contains a set of so-called decomposition methods for each abstract task. Each method is simply a pair mapping an abstract task to a plan, which is a partially ordered set of further primitive or abstract tasks. A planning problem then consists of such an initial plan that needs to be refined into an executable sequence of actions [21], [19].

For representing conceptual knowledge, we consider an ontology formulated in OWL.[2] For the described approach, the restricted EL profile of OWL2 is sufficient, but more expressive languages could as well be used (e.g. OWL2 DL). OWL is underpinned by description logics, whose syntax we adopt in the following. As usual, concept names are denoted with capital letters $A, B, C, \ldots$; role names with small letters $r, s, \ldots$; individuals with small letters $a, b, \ldots$; and the universal concept with $\top$. Complex concept expressions are formed by using conjunction ($C_1 \sqcap C_2$) and existential restriction ($\exists r.C$). Axioms that specify the subconcept relationship between two concept expressions C_1 and C_2, also known as subsumption, are denoted as $C_1 \sqsubseteq C_2$. Concept assertion axioms assert that an individual a is in the extension of a concept (expression) C, written $C(a)$, and role assertions specify that two individuals a, b are connected by role r, written as $r(a, b)$. Several further constructors are included in OWL2 EL (e.g. equivalence and disjointness axioms for concepts, role composition and role inclusion).[2] Description logics have a set-theoretic semantics, where an interpretation $\mathcal{I}$ assigns concepts and individuals to sets and elements, respectively, in a domain Δ and roles to binary relationships over the domain. An interpretation $\mathcal{I}$ is a model of an ontology $\mathcal{O}$ (a set of axioms) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$). An axiom α is entailed by an ontology ($\mathcal{O} \models \alpha$) if all models of $\mathcal{O}$ also satisfy α. Ontology reasoners provide services to reason over ontologies, e.g., they can be used to determine entailments (such as concept subsumptions) of an ontology.

V. A Paradigm for Coupling Procedural and Conceptual Knowledge

In complex *companion systems*, a multitude of components is necessary to provide the offered assistance services. In our use case – assisting humans in handling complex mechanical appliances and tools – this includes at least (cf. Fig 1):

- a planning component – determining the steps the user should take in order to achieve his goal and answering procedural questions to the user
- a knowledge management component – handling the factual knowledge of the domain and answering factual questions to the user
- a dialogue component – mediating the interaction between companion system and user

Since all these three components use model-driven approaches, we have to create an adequate model for each of them.

[1] Abstract tasks are also referred to as compound in HTN planning [21].

[2] Web Ontology Language, see https://www.w3.org/TR/owl2-profiles/

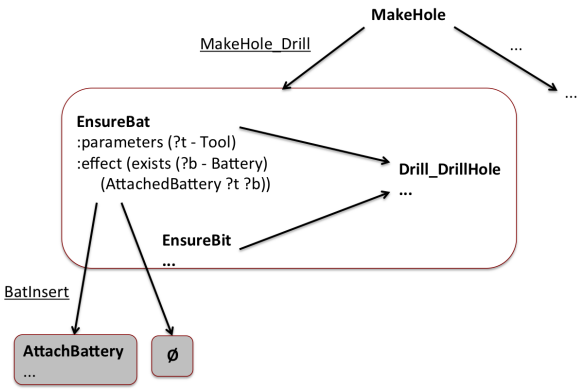

Fig. 2. Part of the decomposition hierarchy for the running example

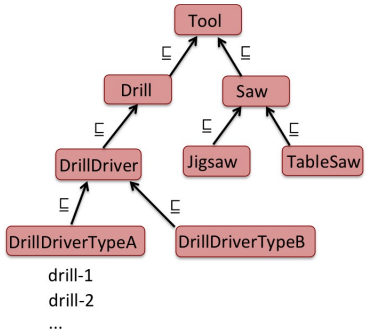

Fig. 3. Part of the concept hierarchy for the running example

However, doing so is a complex process (requiring expert-level knowledge in several disciplines) and introduces natural redundancies between the models. For example, both the models of the planning and the knowledge management component have to contain possible configurations for each tool, e.g. the information which battery fits into which device. The planning component has to find suitable configurations for each tool and determine the order in which they should be used, while the knowledge base has to be able to explain why only a certain configuration is allowed (e.g. for drilling into softwood, or why a 16V battery cannot be used with a 9V device). Such double-specifications are hard to create and even more difficult to maintain over time – especially, if the model has to be changed or extended. Therefore, the overall design objective is to minimise redundancy by specifying each fact only once in the best-suited model for dealing with it. The necessary facts for the other models are generated automatically from the single source model.

In this paper, we focus mainly on the interaction between the planning model and the knowledge base, as their connection is the most complex one. Using this design principle, we have to specify procedural knowledge – how to do things, which effects tasks have and how they depend on each other – in the planning model, while the ontology handles factual and conceptual knowledge. For the running example of drilling a hole with an electric drill, consider the part of the planning domain depicted in Fig. 2. Tasks are written in bold (e.g. *MakeHole*), primitive tasks are in bold surrounded by filled boxes, and method decompositions are represented by labelled arrows where method names are underlined (e.g. *MakeHole_Drill*). The unlabelled arrows represent dependencies between pre- and postconditions. The graph shows that the *MakeHole* task can be decomposed by the *MakeHole_Drill* method which introduces three subtasks (*EnsureBat*, *EnsureBit* and *Drill_DrillHole*). The EnsureBat task in turn is decomposed (which results in a battery being attached to a tool, as a precondition for drilling) by applying the primitive task *AttachBattery*, which is discussed in more detail later.

Sharing Concepts and Properties

Relationships between objects and concepts are specified in the ontology. This includes, for instance, descriptions of the currently available tools, devices, materials, as well as their properties. By generating the planner's problem description from these assertions, we make it easily accessible to the planner, while retaining a standard semantics for the planner's tasks. Accordingly, we design tasks to only refer to those properties in the domain that are relevant for the applicability of an action, such that the planning domain remains generic w.r.t. the individual objects that are available in a concrete planning problem.

Concepts in the application domain are modelled as concepts/classes in the ontology, and particular instances thereof (e.g. devices) as individuals (cf. Fig. 3). Concepts are arranged in a taxonomy. For instance, suppose that a particular device *drill-1* is an instance of a class of devices $DrillDriverTypeA$. Suppose further that it is either specified in the ontology (as in Fig. 3), or that it can be inferred that all devices $DrillDriverTypeA$ are also of class $Tool$ (subsumption). Then *drill-1* is (by inference) also an instance of $Tool$. Role assertions are used to model binary relationships between individuals in the domain or between individuals and values such as character strings or numbers.

In the planning domain, concepts are represented as sorts while objects are represented as constants of the respective sorts. Before the planner is executed, the information represented in the ontology is transformed into the planner's initial state. Each concept name C in the ontology corresponds to a sort C in the planning model and for every instance t of C (i.e. each constant of sort C), we add the respective assertion, written `t - C` in PDDL syntax [20], to the planning model.

Binary relationships – role assertions – in the ontology represent the connections and relations between the represented objects. More specifically, they represent both the current configuration of objects in the real world, as well as higher-level relationships. This latter category, e.g. includes the knowledge which configurations of a tool are suited for which specific task.

Role assertions are made available to the planner in the initial state as binary predicates. For concrete roles/data properties, we have to add additional sorts (like `Number` or

```
1  (:action AttachBattery
2   :parameters (?t - Tool ?b - Battery)
3   :precondition (and
4     (exists (?tt - Type ?bt - Type)
5       (and (typeOf ?t ?tt) (typeOf ?b ?bt)
6         (exists (?conf - BatteryConfig)
7           (and (master ?conf ?tt) (slave ?conf ?bt)))
8     ))
9     (not (exists (?c - Battery) (AttachedBattery ?t ?c)))
10    (not (exists (?u - Tool) (AttachedBattery ?u ?b))))
11  :effect (AttachedBattery ?t ?b)
12 )
```

Fig. 4. Task declaration *AttachBattery* for the running example

`String`) to the planning model and add the data values occurring in the ontology as instances of these sorts.

By strictly adhering to this separation principle, when writing the planning domain model, we would have no access to the sort definitions, as they are contained in the ontology and only transferred to the planning model when starting a planning process. However, the modeller of the planning domain needs to be able to refer to these sorts where they are relevant for specifying methods and tasks. For instance, the `AttachBattery` task (cf. Fig. 4) can only be performed with a battery (which is a concept in the ontology).

This means that the ontology and the planning model have to agree on a minimal common vocabulary as a kind of interface between them. Ideally, it is as lean as possible. This is achieved by using the most generic concepts that are needed to specify the planning domain (e.g. that `AttachBattery` applies to instances of the `Tool` concept), and allowing more fine-grained concepts in the ontology, where inference can be used to establish all their superconcepts (e.g. that any particular instance of *DrillDriver* is also a *Tool*).

The Case Of N-ary Relations

In complex technical domains, the applicability of an action sometimes depends on n-ary relationships (e.g. whether an action requires a specific combination of device(s) and property values). Ontology languages, however, are typically restricted to binary relations (roles). We reify each n-ary relationship by introducing an instance representing such a "configuration" in the ontology. Each individual parameter of the configuration is then assigned using binary (abstract and/or concrete) roles. Accordingly, the tasks quantify over configurations and their binary predicates, instead of using n-ary predicates. We see an example for such a reified relation[3] in the task declaration shown in Fig. 4: The existence of an instance of a `BatteryConfig` is required as part of the precondition (in line 6), which represents valid combinations of batteries and tools. The roles `master` and `slave` denote that a device can be equipped with a specific battery type. Such configurations are also obtained as part of the initial state by translation from the ontology, with a small twist. In the planning domain, configurations play the role of a special kind of for-all statement: they specify that all instances of the concepts $A, B, C, \ldots$ linked in the ontology by a configuration are valid as part of that specific configuration. For instance, a $BatteryConfig$ in the ontology might specify that a $DrillDriverTypeA$ is compatible with a $BatTypeA$ battery (written in first-order logic): $\forall x, y : DrillDriverTypeA(x) \wedge BatTypeA(y) \rightarrow compatible(x, y)$. Such kinds of statements are known as concept products in description logics [22], but the feature is not part of OWL EL and unsupported in standard reasoners.

To represent such configurations in the ontology, we introduce individuals of a special concept $Config$. Each individual is linked to the concepts that are specified to be part of the configuration using axioms of the form $(\exists r.A)(c)$, where c is the individual and r is a role. E.g., the assertions

$$BatteryConfig(conf1)$$
$$(\exists master.DrillDriverTypeA)(conf1)$$
$$(\exists slave.BatTypeA)(conf1)$$

represent the above-mentioned compatibility between $DrillDriverTypeA$ and $BatTypeA$. Note that the $Config$ concept is subdivided into more specific concepts (e.g. $BatteryConfig \sqsubseteq Config$). When dealing with an individual c representing a $Config$, the associated axioms with existential restrictions are translated to atoms of the form `(r c A)` in the initial state. Thus, in the example, the planning model's initial state receives the declaration `conf1 - BatteryConfig` (and by inference also `conf1 - Config`) and the atoms

```
(master conf1 DrillDriverTypeA)
(slave conf1 BatTypeA)
```

These atoms are then able to fulfil the precondition in lines 6–7 in Fig. 4. Additionally, if $\mathcal{O} \models DrillDriverTypeA(tool1)$ and $\mathcal{O} \models BatTypeA(bat1)$, the initial state will also contain the atoms `tool1 - DrillDriverTypeA` and `bat1 - BatTypeA`. Therefore, the precondition of the `AttachBattery` action, which is responsible for actually inserting a battery into a tool, can determine the sorts (`?tt` and `?tb`) of both the tool `?t` and the battery `?b` using the `typeOf`-predicate.[4] Based on given sorts, it can be determined whether attaching a battery of sort `?tb` to a tool of sort `?tt` is allowed. Since the configuration-related predicates (e.g. `master` and `slave`) are static in the planning model, i.e. they cannot be changed by any action, reification comes at no cost in the planning model – static preconditions are statically evaluated before planning.

VI. Prototype Realisation

A prototype system has been implemented that demonstrates the proposed methodology. Reasoning and planning are performed using the ontology reasoner JFact[5] and the planning

[3] It would not technically be necessary here, because we reify a binary predicate. We have nevertheless chosen this example to keep the illustration of the applied principle as simple as possible.

[4] `typeOf` is an extension of the PDDL standard for general qualifications over sorts. This construct is compiled into standard PDDL.

[5] http://jfact.sourceforge.net/

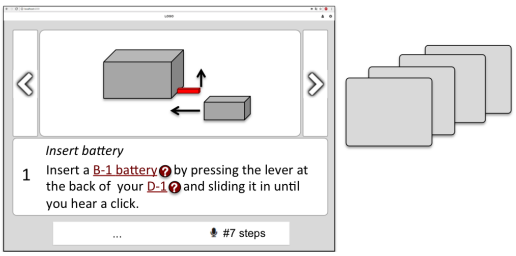

Fig. 5. Interface

system PANDA [23]. Thanks to the described approach, the planner does not need to be equipped with its own description of all the objects and concepts in the domain, as usual in planning, but receives this information dynamically from the ontology component. Since all objects in the planning domain originate from the ontology, it is possible to query the ontology about the concepts, their relationships and additional information (e.g. descriptions and hints). An interface (cf. Fig. 5) is provided to the system by a front-end web application created using the Vue.js framework[6] in combination with a dialogue management module. The dialogue component uses LUIS[7] for speech recognition to identify planning goals. Generated plans are presented in the form of slides and synthesised speech.

VII. Conclusion

Parametrising the planning domain by information represented in the ontology enables the integration of procedural and conceptual knowledge while maintaining a clean separation. As a result, the information in the ontology can be changed (e.g. devices added, properties added, etc.) and becomes dynamically available for planning. A further advantage of the tight integration of conceptual and procedural knowledge comes to bear when a *companion system* is required to deliver explanations for the plans it generates. To-be-explained relationships in the planning domain directly refer to concepts and relations in the ontology, such that both procedural and conceptual knowledge can be combined in the generated explanations.

Acknowledgement

This paper describes ongoing work within the technology transfer project "Do it yourself, but not alone: Companion Technology for Home Improvement" of the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG). The industrial project partner is the Corporate Research Sector of the Robert Bosch GmbH. We would like to thank the anonymous reviewers for their valuable feedback.

References

[1] S. Biundo and A. Wendemuth, "*Companion*-technology for cognitive technical systems," *Künstliche Intelligenz*, vol. 30, no. 1, pp. 71–75, 2016, special Issue on Companion Technologies.

[2] S. Biundo, D. Höller, B. Schattenberg, and P. Bercher, "Companion-technology: An overview," *Künstliche Intelligenz*, vol. 30, no. 1, pp. 11–20, 2016, special Issue on Companion Technologies.

[3] P. Bercher, S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg, "Plan, repair, execute, explain - How planning helps to assemble your home theater," in *Proc. of ICAPS*. AAAI Press, 2014, pp. 386–394.

[4] P. Bercher, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nothdurft, F. Honold, W. Minker, M. Weber, and S. Biundo, "A planning-based assistance system for setting up a home theater," in *Proc. of AAAI*. AAAI Press, 2015, pp. 4264–4265.

[5] G. Behnke, D. Ponomaryov, M. Schiller, P. Bercher, F. Nothdurft, B. Glimm, and S. Biundo, "Coherence across components in cognitive systems – One ontology to rule them all," in *Proc. of IJCAI*. AAAI Press, 2015, pp. 1442–1449.

[6] Y. Gil, "Description logics and planning," *AI Magazine*, vol. 26, no. 2, pp. 73–84, 2005.

[7] S. Ahmetaj, D. Calvanese, M. Ortiz, and M. Šimkus, "Managing change in graph-structured data using description logics," in *Proc. of AAAI*. AAAI Press, 2014, pp. 966–973.

[8] A. A. Sánchez-Ruiz, P. A. González-Calero, and B. Díaz-Agudo, "Abstraction in knowledge-rich models for case-based planning," in *Case-Based Reasoning Research and Development*, vol. 5650. Springer, 2009, pp. 313–327.

[9] E. Sirin, "Combining description logic reasoning with AI planning for composition of web services," Ph.D. dissertation, University of Maryland at College Park, 2006.

[10] R. Hartanto and J. Hertzberg, "Fusing DL reasoning with HTN planning," in *KI 2008: Advances in AI*, ser. LNCS, vol. 5243. Springer, 2008, pp. 62–69.

[11] ——, "On the benefit of fusing DL-reasoning with HTN-planning," in *KI 2009: Advances in AI*, ser. LNCS, vol. 5803. Springer, 2009, pp. 41–48.

[12] A. Freitas, D. Schmidt, A. Panisson, F. Meneguzzi, R. Vieira, and R. H. Bordini, "Semantic representations of agent plans and planning problem domains," in *Proc. of EMAS 2014*, ser. LNCS, vol. 8758. Springer, 2014, pp. 351–366.

[13] A. V. Feljan, A. Karapantelakis, L. Mokrushin, R. Inam, E. Fersman, C. R. B. Azevedo, K. Raizer, and R. S. Souza, "KMARF: A framework for knowledge management and automated reasoning," in *Proc. of ISEC 2017 Workshops: ModSym, DIAS, and EDUDM*, 2017, vol. 1819, CEUR Workshop Proceedings.

[14] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, "EasyLiving: Technologies for intelligent environments," in *Proc. of Handheld and Ubiquitous Computing*, ser. LNCS, vol. 1927. Springer, 2000, pp. 12–29.

[15] E. Kaldeli, E. U. Warriach, A. Lazovik, and M. Aiello, "Coordinating the web of services for a smart home," *ACM Trans. Web (TWEB)*, vol. 7, no. 2, pp. 10:1–10:40, 2013.

[16] I. Georgievski, T. A. Nguyen, and M. Aiello, "Combining activity recognition and AI planning for energy-saving offices," in *Proc. of UIC/ATC*. IEEE, 2013, pp. 238–245.

[17] B. Krieg-Brückner, S. Autexier, M. Rink, and S. G. Nokam, "Formal modelling for cooking assistance," in *Software, Services, and Systems*. Springer, 2015, pp. 355–376.

[18] P. Bercher, D. Höller, G. Behnke, and S. Biundo, *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Springer, 2017, ch. User-Centered Planning, pp. 79–100.

[19] ——, "More than a name? On implications of preconditions and effects of compound HTN planning tasks," in *Proc. of ECAI*. IOS Press, 2016, pp. 225–233.

[20] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *JAIR*, no. 20, pp. 61–124, 2003.

[21] T. Geier and P. Bercher, "On the decidability of HTN planning with task insertion," in *Proc. of IJCAI*. AAAI Press, 2011, pp. 1955–1961.

[22] S. Rudolph, M. Krötzsch, and P. Hitzler, "All elephants are bigger than all mice," in *Proc. of DL*, ser. CEUR Workshop Proceedings, vol. 353, 2008.

[23] P. Bercher, S. Keen, and S. Biundo, "Hybrid planning heuristics based on task decomposition graphs," in *Proc. of SoCS*. AAAI Press, 2014, pp. 35–43.

[6]https://vuejs.org/ [7]https://www.luis.ai/