Alice in DIY wonderland or: Instructing novice users on how to use tools in DIY projects

Gregor Behnke [a,*], Marvin Schiller [a,c], Matthias Kraus [b], Pascal Bercher [a], Mario Schmautz [a], Michael Dorna [c], Michael Dambier [c], Wolfgang Minker [b], Birte Glimm [a], and Susanne Biundo [a]
[a] *Institute of Artificial Intelligence, Ulm University, Germany*
E-mails: gregor.behnke@uni-ulm.de, marvin.schiller@alumni.uni-ulm.de, pascal.bercher@uni-ulm.de, mario.schmautz@uni-ulm.de, birte.glimm@uni-ulm.de, susanne.biundo@uni-ulm.de
[b] *Institute of Communications Engineering, Ulm University, Germany*
E-mails: matthias.kraus@uni-ulm.de, wolfgang.minker@uni-ulm.de
[c] *Corporate Research Sector, Robert Bosch GmbH, Renningen, Germany*
E-mails: marvin.schiller@de.bosch.com, michael.dorna@de.bosch.com, michael.dambier@de.bosch.com

Abstract. We present the interactive assistant ROBERT that provides situation-adaptive support in the realisation of do-it-yourself (DIY) home improvement projects. ROBERT assists its users by providing comprehensive step-by-step instructions for completing the DIY project. Each instruction is illustrated with detailed graphics, written and spoken text, as well as with videos. They explain how the steps of the project have to be prepared and assembled and give precise instructions on how to operate the required electric devices. The step-by-step instructions are generated by a hierarchical planner, which enables ROBERT to adapt to a multitude of environments easily. Parts of the underlying model are derived from an ontology storing information about the available devices and resources. A dialogue manager capable of natural language interaction is responsible for hands-free interaction. We explain the required background technology and present preliminary results of an empirical evaluation.

Keywords: Companion-technology, Digital assistant, Planning-based assistance, Knowledge representation

1. Introduction

Imagine a novice user who has never used electric tools in her life before. Let's call her Alice. She has, so far, relied either on pre-built furniture or on craftsmen or friends doing constructions for her. Alice, however, wants to become more self-reliant and wants to learn how to perform small household constructions and repairs, or simply hobby projects like a key rack or a bird nest box by herself. Such activities are commonly called do-it-yourself (DIY) projects.

Even the realisation of simple DIY projects often requires the usage of electric tools of different kinds: drills, saws, or sanders are just some of the very commonly used tools, in particular when working with wood – an important material in DIY projects. Without proper instruction on how to use such devices, they may be difficult to use or even dangerous for novice users. Alice might spend a long time to figure out how to handle and configure these tools for the current task at hand. Possible dangers, like accidents with an electric saw, might prevent novice users from starting to use them in the first place.

We developed the interactive assistant ROBERT (named after Robert Bosch GmbH) that provides appropriate instructions for the realisation of DIY projects. Alice is walked through the steps required for completing a DIY project, which are illustrated with detailed graphics, text, and videos. The provided explanations include both project-specific as well as project-independent aspects: That is, they explain how parts of the given project are to be assembled or prepared, but they also give detailed instructions on how the required electric devices are to be handled. This enables Alice to complete her project successfully while simultaneously learning how to use the employed tools

*Corresponding author. E-mail: gregor.behnke@uni-ulm.de.

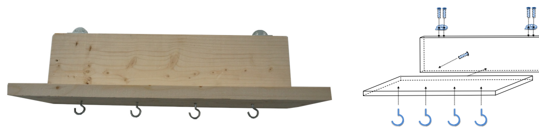


Fig. 1. A real key rack and its conceptual drawing [1].

safely, enabling her to complete DIY projects independently in the future. ROBERT is designed to be generic such that additional DIY project descriptions can be added solely via additions to the underlying knowledge models. Those parts of the model describing general, project-independent information (e.g. specifications on how to use the electric devices, how to connect two wooden boards, or information about the properties of materials) and media content are (re-)used for new projects. To exemplify our methodology, we have instantiated our assistant ROBERT with a project realising the construction of a key rack (an assembled key rack can be seen in Fig. 1).

In this article, we demonstrate ROBERT (illustrated using the key rack project) and its underlying technology. ROBERT smoothly combines and integrates three core capabilities: hierarchical planning, ontological reasoning, and dialogue management. Each of these capabilities is based upon a formal model of the DIY domain. A focus of our work lies on providing appropriate integration techniques between these models. By clearly and cleanly dividing the required knowledge between the models we achieve both an efficient knowledge integration as well as avoid redundancy among the models. In this article we will especially describe the means by which we transfer required information between the individual components. The underlying architecture is *independent* of the deployed application scenario, so that assistants for various other application scenarios can be realised by suitably extending the required models and media data. Based on ROBERT's formal models a *plan* is computed, i.e. a sequence of actions. The user is instructed to follow these actions to achieve the given task such as, in our example, building a key rack. We pursue a *hierarchical* planning approach [2–4], where the given task is refined step-wise into more primitive courses of action until a completely primitive, i.e. directly executable, plan is generated. This way, instructions can be presented at different levels of abstraction.

The technology behind ROBERT is based on our experience with an earlier research prototype assistant, providing support for setting up a complex home theatre [5–9]. Both assistants – our novel DIY assistant as well as our former assembly assistant – base upon a generic system architecture that allows for realising assistants for a broad variety of tasks. Due to our experience with the earlier prototype, the requirements for an industrial application of ROBERT, and the requirements imposed by the DIY domain, we have developed several new techniques for ROBERT. ROBERT uses an ontology to store and retrieve factual domain knowledge, instead of a database-like knowledge management. Further, ROBERT handles the task hierarchy fully inside the planner – again in contrast to the previous assistant – and communicates the hierarchical structure of a plan to the dialogue manager. This allows the hierarchy to express actual choices between alternative courses of action. This allows us, e.g. to present instructions corresponding to abstract tasks in the planning model. Further, we can now use the hierarchy to enhance the dialogue, e.g. by structuring the shown steps visually and by providing feedback to the user once a section of the instructions has been completed.

ROBERT utilises an ontology and its reasoning capabilities for knowledge representation [10] instead of relying on a purely database-like system. This enables easy scalability and knowledge maintenance. That way we can exploit existing ontologies storing information relevant to the application domain. In our application scenario, this information pertains to electric devices and DIY resources. Reasoning further allows for inferring certain properties automatically. For example, under the assumption that every drill bit for metal can also be used for drilling in wood, it is inferred that any particular metal drill bit (e.g. one of 3 mm diameter) can be used in place of a wood drill bit of the corresponding size.

Another benefit from integrating an ontology and its reasoning capabilities are improved explanation capabilities. Inferences made by the system are made available to the user in the form of natural language text. For example, in ROBERT's ontology it is formalised that if a screw is small enough, it can be driven into softwood without pre-drilling (otherwise, the existence of a pre-drilled hole is a precondition for the screwing action). Based on this, the system can infer in what situations pre-drilling is necessary, and can deliver a *verbalisation* of the reasoning steps that were applied. Consider, for instance, the activity of driving a screw of 3 mm diameter into a plank made of spruce. Using the system's taxonomy of materials and formalised properties, it can be inferred that pre-drilling is not necessary. Furthermore, the reasons (an explanation) can be made explicit on request why "screwing a 3 mm screw into spruce does not require pre-drilling":

Spruce is softwood, thus screwing a 3 mm screw into spruce applies to softwood. Since a 3 mm screw is a screw that has a diameter of 3 mm, by definition it is a small screw. Hence screwing a 3 mm screw into spruce is done with a small screw. Screwing a small screw into softwood does not require pre-drilling. Therefore, screwing a 3 mm screw into spruce does not require pre-drilling.

Such explanations are not pre-formulated, but generated at runtime from the available facts in the ontology from a formal proof for the relationship in question. Explanations enable Alice to understand the system's reasoning and its behaviour and to learn about the application domain, i.e. DIY and electric tools, while using the system. A challenge with automated explanations, however, are technical details that may seem too obvious to Alice but which are logically necessary for the system.

The paper is structured as follows. We first introduce relevant theoretical and technological concepts in Sec. 2 and then review related systems and technology in Sec. 3. In Sec. 4, we give a high-level overview of our system by explaining it from Alice's view. In Sec. 5 we explain the *system architecture*, i.e. its components and how they interact. In Sec. 6 we explain how we maintain the different kinds of knowledge required by the system and how it is conveyed to the user. Sec. 7 explains the underlying planning model and how its hierarchy is exploited for illustration purposes. Sec. 8 is about the *dialogue management*, whose primary responsibility is handling user input and output. Finally, we present preliminary results of a user study with our prototype system in Sec. 9 before concluding the paper in Sec. 10.

2. Preliminaries

We next introduce the basics of the employed planning and knowledge modelling formalisms.

2.1. Planning

ROBERT uses planning to determine the instructions presented to its user Alice. Its planning domain is formulated using Hierarchical Task Network (HTN) planning [2, 4]. It distinguishes two types of actions: primitive actions A and abstract tasks T. Primitive actions $a \in A$ can be executed directly, i.e. by the user Alice, without the need to separate them into even more basic steps, e.g. pushing a button of an electric drill. *Abstract* tasks on the other hand describe more complex courses of action, e.g. connecting two pieces of wood with screws, which must be refined into more concrete actions and usually also offer variability in the ways they can be executed.

Each action a is described in terms of a pair of preconditions and effects $\langle p, e \rangle$. The precondition is a function-free first order formula, which must be true in the state prior to the action being executed. States are expressed using lifted state predicates. Each state s is a set of instantiations of these predicates which are considered true. An effect is a list of function-free literals (i.e. a positive or negative predicate with parameter variables), which are divided into positive (adding) effects p^+ and negative (deleting) effects p^-. If an instantiation of a, i.e. a grounding of a with constants as its arguments, is executed in a state s in which its precondition holds, the resulting state is defined as $(s \setminus p^-) \cup p^+$. Technically, the planner uses a format based on PDDL [11, 12] extended for hierarchical planning domains. We show in Fig. 7 an example of a primitive action that is part of ROBERT's domain model. In PDDL we declare with the statement `o - T` that the object `o` belongs to the type `T`. PDDL uses the syntax `(p o`$_1$` ...o`$_n$`)` to declare that the fact `p(o`$_1$`, ..., o`$_n$`)` is true.

The connection between primitive actions and abstract tasks is made in HTN planning via decomposition methods. They specify a grammar-like refinement structure for abstract tasks. This structure is given in terms of decomposition methods (t, tn), consisting of a task t to be refined and a task network tn specifying the result of the refinement, i.e. an allowed means for achieving t. Task networks are the central element of HTN planning. A task network tn is a partially ordered set of instances of primitive actions and abstract tasks. Formally, a task network is a tuple $(L, \prec, \alpha)$, where L is a set of labels, $\prec \subseteq L \times L$ is a strict partial order[1] on L, and $\alpha : L \to A \cup T$ is a function that assigns to each label its primitive action or abstract task. The introduction of labels is necessary to be able to represent task networks containing the same task more than once. If a decomposition method (t, tn) is applied to an abstract task t in a task network tn', we replace t by the contents of tn [4]. If there exists any sequence of decomposi-

[1]A strict partial order is an irreflexive, antisymmetric, and transitive relation.

tions transforming a task network *tn* into another tn', we write $tn \rightarrow_D^* tn'$.

The goal in HTN planning is described in terms of both a state-based goal formula g and a task network tn_I, called the initial abstract plan. A solution of the planning problem is a sequence of primitive actions π, which is executable in the initial state and must reach a state in which g is true. The individual actions of such a sequence π are called plan steps. As a further requirement, this sequence of actions π must be a refinement of the initial abstract plan tn_I [4]. This allows for specifying high-level activities that are to be performed in the initial abstract plan, e.g. building a key rack. Formally, there must be a refinement tn' of tn_I, i.e. $tn_I \rightarrow_D^* tn'$, such that π is a linearisation of tn' (a linearisation of tn' is any topological ordering of its tasks that is compatible with the ordering constraints in tn').

2.2. Ontology-based knowledge modelling and verbalisation

Declarative (factual) domain knowledge employed by ROBERT is stored in an ontology. The ontology is used to formalise DIY concepts (e.g. classes of tools and their properties) and to store associated information such as instruction texts and references to images and videos. This knowledge is formulated in the ontology language OWL2[2], whose main constructs relevant for the remainder of the paper are introduced in the following. In this paper, we employ the notation of description logic (cf. [13]), which provides a semantics for OWL2. We denote concepts by capital letters *A,B,C,...* and use camel case for specific concept names (e.g. *DrillDriver*). The universal concept is denoted by $\top$ and the unsatisfiable (empty) concept by $\bot$. Individuals are denoted by small letters *a,b,....* Membership of an individual a in a concept A is stated in the form of a concept assertion $A(a)$ and we say that a is an instance of A. So-called (object) properties or (abstract) roles are denoted by small letters *r,s,...* and express relationships between pairs of individuals, e.g. the property assertions $r(a,b)$ expresses that the individual a is r-related to the individual b. The semantics of description logics is defined model-theoretically; concepts are interpreted as subsets of a domain, properties as binary relations, and individuals as elements of the domain. Constructors can be used to form complex concept expressions: conjunction and disjunction are written as $\sqcap$ and $\sqcup$, respectively, and negation is written as $\neg$. The so-called existential restriction $\exists r.C$ specifies the concept whose instances are related by the property r to some instance of the concept C. For example, the concept description $\exists hasVisualFeature.CrossShape$ specifies all instances of the domain that are related to some *CrossShape*. The universal restriction $\forall r.C$ denotes the concept whose instances are only related by r to instances of C, if at all. The statement that a concept A is a taxonomical sub-concept of another concept B is referred to as a *subsumption axiom*, written as $A \sqsubseteq B$. Equivalence is mutual subsumption between concepts and denoted by $\equiv$. Among further constructors (which we do not detail here), OWL2 provides the specification of attribute/value pairs in the form of data properties (also referred to as *concrete* roles), for which we use the syntax $\exists p =_{\text{“value”}}$.

Traditionally, ontologies are split into a terminological (conceptual) part called TBox, and an assertional part (ABox) where concept assertions and property assertions are specified for individuals. An interpretation $\mathcal{I}$ is called a model of an ontology $\mathcal{O}$, iff it satisfies all axioms in $\mathcal{O}$. An axiom is entailed by an ontology $\mathcal{O}$ if all models of $\mathcal{O}$ also satisfy this axiom. Ontology reasoners offer reasoning services for ontologies, for instance they can be used to determine the entailed axioms of an ontology. The discipline of *ontology verbalisation* is concerned with the production of natural language text from knowledge contained in and/or inferred from ontologies.

3. Related work

Planning-based assistance systems have been developed in the past. However most of them focus on users that are experts in the domain for which the assistant is designed. The PASSAT system is a mixed-initiative planning system designed for collaboratively creating plans with a human user [14]. Its objective is to create plans for military operations in cooperation with the commanding officer which are executed by military personnel. Like our system, the basis of PASSAT is completely domain-independent, and the authors rely solely on a model of the specific application domain for the domain in question. PASSAT uses a hierarchically modelled domain, though only primitive plans are visualised after successful plan generation in a step-by-step fashion. Notably it lacks the combination with an ontology to store and manage its background knowledge – which is instead hard-coded into the planning

[2]https://www.w3.org/TR/owl2-overview/

model – nor does PASSAT use advanced dialogue capabilities to mediate the interaction with the user.

MAPGEN [15] is a planning-based assistant developed by NASA to assist scientists and spacecraft operators in scheduling activities for the Mars rovers. It is capable of changing the plan to include specified activities and resolves resulting conflicts between activities. However, its UI is designed for efficiency (it is essentially a complex Gantt chart) with expert users in mind.

Two assistants exist addressing the emergency & firefighting domain. RADAR [16] develops a firefighting and rescue plan together with its expert user, e.g. a fire-brigade chief. RADAR uses a classical, i.e. non-hierarchical, domain and does not combine its capabilities with an ontology. Actions are presented only by their names, while emphasis in the UI is laid on geographical information and the availability of vehicles and materials. As such, the system heavily relies on an expert user being able to understand what the actions mean. The SIADEX system provides decision support for crisis management in a forest-fire fighting domain [17]. They also employ a hierarchical planning framework and exploit ontologies in which the required knowledge is stored. Solution plans may also be displayed to non-experts, but rather than showing them in a step-by-step fashion, they use Microsoft Excel chronograms and Microsoft Project Gantt charts.

In contrast, only a few systems assist "novice" users and try to teach them on their application domain. One of them is an assistant for elderly people and people with cognitive impairments [18]. It helps these people to remember and execute routines of their daily lives in order to allow them to retain their independence for a longer time. Routines include, among others, using the bathroom, taking medicine, eating and drinking, and housekeeping. In contrast to our system, it employs a model of time (which is particularly important for maintaining routines of daily life), but it does neither feature a task hierarchy nor explanations. The SHIP-tool [19] is designed to assist in planning and preparing meals. It uses description logic directly to describe states. This highly expressive language prohibits SHIP from using modern domain-independent planning systems and instead requires a specialised planner that can handle such states. In contrast, ROBERT translates the knowledge stored in its ontology into a purely propositional representation enabling it to use highly efficient search-based planners.

Further, Steinberger et al. [20] propose a cognitive assistance architecture incorporating a "semantic" manual (using an ontology) to model instruction steps. However, these steps are neither hierarchical, nor is a planning system being used.

There are several related works in the area of ontology verbalisation and the explanation of inferences. Several approaches have been proposed to generate descriptions of concepts and individuals contained in an ontology (e.g. [21–23]) and to make inference steps explicit (cf. [24–27]). This is done for the benefit of users who are not familiar with OWL or description logic syntax, and the explanation text presented in the introduction provides an example of such a generated text. The automated verbalisation of ontology content begs the question how well these synthetic texts are understood by untrained readers. Therefore, the development of these approaches has been accompanied by empirical studies focusing on different aspects of the generated explanations. The text quality of generated descriptions has been studied by Androutsopolous et al. [23], who present a comparison of their own NaturalOWL system with the more generic SWAT verbaliser [21]. The experiments were carried out with computer science students who had to rate the texts for fluency, referring expressions, text structure, clarity and interest. The experiments confirmed the authors' hypothesis that the use of text planning, together with domain-dependent generation resources for sentence planning, aggregation and referring expressions has a measurably positive effect on perceived text quality.

The verbalisation of inference steps has been empirically studied by Nguyen et al. [26, 28] in online experiments. They found that depending on the kind of employed inference rule, the generated explanations were more or less likely to be accepted or rejected by the participants. Whereas rules that straightforwardly connect subsumptions, exploit equivalence, or split or combine conjunctions (an example is shown in Sec. 6.4) were generally accepted by participants, the application of some inference rules appeared less understandable to participants (e.g. some rules involving contradiction/unsatisfiability). Using these results for single inference rules, the authors were able to predict the understandability of derivations using two inference steps (cf. [28]).

The understandability of longer explanations generated from derivations was also examined empirically. Schiller et al. [27] provide some first empirical evidence that the shortening of explanations by hiding and combining inference steps considered obvious does not negatively affect understandability. In another study [29], explanations generated by the mechanism employed in this paper were tested by applying a psychological model of text comprehension to it. This helped to identify and rectify cases in which textual coherence was found to be deficient. The understandability and quality of the resulting texts was studied in an online study reported in [29]. The two abovementioned experiments have not used the DIY domain (as in the experiment reported in Sec. 9), but had a more general focus (in case of [27], the TONES[3] collection of ontologies). Therefore, the experiment presented here can be considered a first exploratory step towards further experiments focusing specifically on the kind of descriptions and explanations employed by ROBERT.

4. Alice's view on ROBERT

Before we describe the techniques used in ROBERT and how its components interact with each other to provide assistance, we describe how a user of ROBERT, i.e. Alice, would interact with it. Interaction with ROBERT starts whenever a user decides that she wants to implement a do-it-yourself project. Initially, ROBERT presents Alice with a list of available projects (e.g. building a key rack, a bird house, a table, refurbishing an old door, or a chair). After selecting a project, Alice informs ROBERT about the tools and materials she possesses. Based on this information, ROBERT uses its planner to generate a sequence of actions – a plan – that can be performed with the available tools and materials and that serves to complete the project requested by the user. Here the strength of planning comes into play. ROBERT does not simply churn out a hard-coded list of instructions for each project, but automatically decides which action can and has to be performed using the actually available tools and materials in order to successfully finish the project Alice wants to complete. In order to generate the plan, ROBERT uses both its planning model as well as factual knowledge about tools and their configurations stored in ROBERT's ontology.

Once the planner has generated a plan, it is transferred to the dialogue manager for presentation. The plan is shown in the form of step-by-step instructions, where each action corresponds to one instruction. Each action is explained to the user by showing her a slide consisting of a textual description, an image, and a video of what to do (cf. Fig. 2). We use the knowledge in ROBERT's ontology to find appropriate media content (cf. Sec. 6). It stores both textual descriptions of individual actions as well as images and videos of how to perform actions. However, as ROBERT can adapt to a large variety of situations it might not posses perfectly fitting media material for the action at hand. For example, ROBERT might instruct Alice to use the PSR18Li2[4] to drill a 15 mm Torx screw into softwood, as ROBERT knows Alice owns this drill, but only has a video of drilling a generic screw using a generic drill into softwood. The available video is still a more appropriate instruction than presenting no media content at all. ROBERT finds the best fitting instructional material via ontology reasoning using the actions and their parameters (i.e. constants). This enables easy scalability when new tools or materials are added and allows for a modular creation of media content.

Alice might also inquire about additional information for every action-based instruction given to her. For example, she might be instructed to insert a clean-for-wood sawing blade into a saw, while not knowing how to recognise this specific type of blade. Here, ROBERT can provide descriptions of objects – including their visual features, solely based on the information stored inside the ontology.

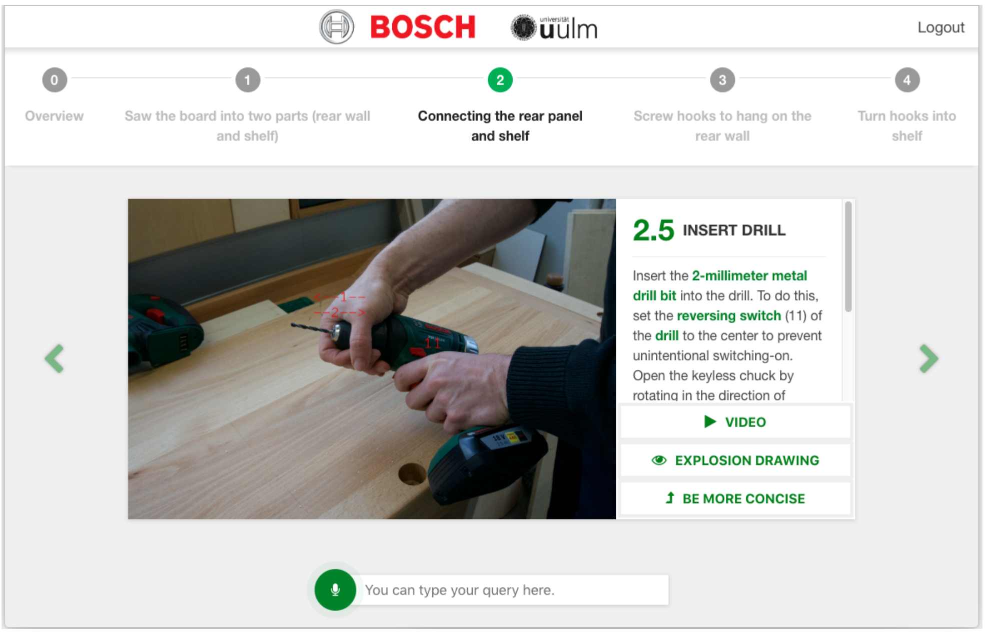
In addition to the detailed step-by-step instructions, ROBERT also generates an abstraction of the presented plan by using the underlying task hierarchy. Since the generated abstraction also contains tasks, we can use ROBERT's ontology to retrieve a textual description and media content for them. While the user is presented the full plan, we show the abstraction as an overview bar at the top of the screen. Here we mark the position of the currently shown action in the abstract sequence. This way Alice knows roughly at which point she is in executing the project and how much of the project is left to do, i.e. she can keep track of her progress. The bar also serves as an instant feedback and reward for the user leading to increased motivation when performing the project.

5. System architecture

ROBERT consists of four major software components: the UI, the planner, the ontology manager, and the dialogue manager. Each of these components is described in detail in the following sections. In this section, we describe which component of ROBERT serves

[3]https://zenodo.org/record/32717

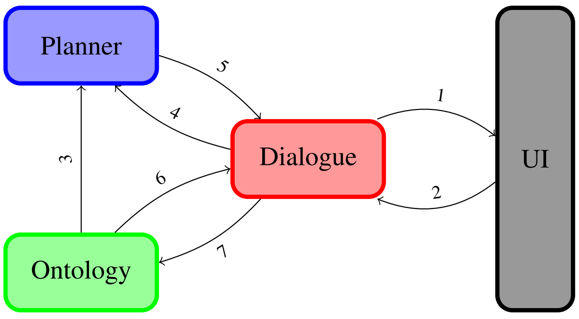
[4]A PSR18Li2 is a model of drill driver aimed at the DIY hobbyist.


BOSCH
uulm
Logout
0 Overview
1 Saw the board into two parts (rear wall and shelf)
2 Connecting the rear panel and shelf
3 Screw hooks to hang on the rear wall
4 Turn hooks into shelf
VIDEO
EXPLOSION DRAWING
BE MORE CONCISE
You can type your query here.

2.5 INSERT DRILL

Insert the **2-millimeter metal drill bit** into the drill. To do this, set the **reversing switch** (11) of the **drill** to the center to prevent unintentional switching-on. Open the keyless chuck by rotating in the direction of

Fig. 2. Instructions for inserting a metal drill bit.



1 display instructions
2 user input
3 initial state
4 planning request, chosen project
5 plan
6 media
7 tasks, available tools, and materials

Fig. 3. ROBERT's system architecture. Each arrow describes data or instructions passed between the individual components. It is roughly based on the work by Bercher et al. [5, 7].

which function, which component stores which information and how these components interact to deliver ROBERT's assistance. Fig. 3 depicts ROBERT's general architecture and shows which information is exchanged between the components. It is roughly based on the architecture employed in our previous assistant for setting up a HiFi system [5, 7]. All components are implemented as web-services exchanging information and instructions as JSON-formatted data [30].

ROBERT's abilities are systematically separated and delivered by the component best suited for them. The planner handles procedural knowledge and generates goal-directed instructions, i.e. plans. The model is formalised as an HTN planning domain [2]. The planner PANDA [31] is employed for generating plans while interaction with other components and further necessary computations, e.g. for computing an abstract plan, are done by a specialised software component. Details on the planner are described in Sec. 7.

Factual knowledge is stored in the ontology, as are references to all media contents. The ontology is written in OWL2 (cf. Sec. 2.2) and reasoning is done by the reasoner HermiT [32]. The ontology manager also handles requests for factual explanations and uses reasoning to retrieve stored media contents. How the ontology manager stores and processes its information is detailed in Sec. 6.

The dialogue manager (Sec. 8) is tasked with enriching the generated plan with media provided by the ontology and transferring it to the user interface. It also handles the user's inputs and decides which component should perform which task as a reaction to the input. The dialogue manager is specifically written for ROBERT, but uses Microsoft Window's built-in speech recognition and Microsoft's LUIS [33] for intent recognition. This enables hands-free natural-language interactions with ROBERT, which is especially important in a DIY-environment, where users often have dirty hands or wear protective gloves and speech-based interaction is the most convenient.

This separation of knowledge leads to an increased need for communication between the components, especially between planner and ontology. It, however, eliminates the need to store information in two components redundantly, making ROBERT easy to maintain and to adapt to new types of projects, new materials, and new devices. For example, static knowledge about materials and tools, e.g. how hard a specific type of wood is or which switches a specific drill has, is not stored inside the planning model, but in the ontology. The planner has to take this information into account when planning. It requests this information from the ontology manager when needed, i.e. when it receives a planning request from the dialogue component. It transforms the received information into a list of objects and facts concerning them. These facts form the initial state of the planning process. The ontology is also responsible for storing the current state of the world. This distribution is systematic, as the current state of the world is not procedural, but a time-dependent and otherwise static information. The media contents themselves are stored in a directory structure which is neither part of the dialogue manager nor the UI. Whenever media is needed, the dialogue manager initiates a query to the ontology, which responds with the path to fitting content.

6. Knowledge management

An instruction, solely based on an action sequence and its abstraction ("How is something done?") is not adequate, as novice users require further information about tools and materials ("How can I identify this drill?"). To provide coherent assistance, the knowledge used by the planner and conveyed in explanations must be in sync [34, 35]. We achieve this by separating factual knowledge and procedural knowledge. Procedural knowledge is stored in the planning model (which is further discussed in Sec. 7), while factual knowledge is stored in a separate ontology. When one of the system's components needs information, it is passed on in an appropriate format. For example, when the planner is searching for a plan involving drilling, the planner's work includes checking if the combination of materials used, the drill's configuration (battery and inserted drill bit) and settings (e.g. suitable rotation speed per type of wood) is appropriate for instantiating the corresponding task. Such factual knowledge is stored in the system's ontology and made available for planning.

The ontology models the following aspects of the DIY domain:

- A taxonomy of DIY concepts is modelled in the TBox. This includes the formalisation of power tools and their properties, together with attachments (e.g. screw bits) and other materials (e.g. screws, types of wood, etc.). The concepts are used in subsumption axioms, e.g. *PSR18Li2*$\sqsubseteq$*Drill-Driver* (as in line 1 in Fig. 4a) states that any instance of the concept *PSR18Li2* is also an instance of the concept *DrillDriver*. Properties of concepts are assigned using (object and data) properties. For example, line 4 in Fig. 4a states that Philips screw bits have a cross shape as a visual feature and line 6 describes a concept defined by the value of a data property.
- Concrete objects (instances of their respective concepts) that are available. These instances are modelled using the ontology's (assertional) ABox. For instance (cf. Fig. 4b), if the ABox specifies that there exists an instance *drill-1* of the class *DrillDriver* (expressed as *DrillDriver(drill-1)*), this instance can be used in planning (and the respective information is passed on to the planner as part of the initial state). The state of individual objects prior to planning is also represented, for instance whether a battery is loaded.
- Relationships that represent valid combinations of tools, materials and settings in line with DIY know-how. These combinations are treated as factual – for instance, consider that for establishing a screw connection in softwood with a 4 mm screw, pre-drilling with 3 mm diameter and a moderate speed setting is appropriate. This information is passed on to the planner, where the existence of such valid combinations is used as a condition for executing the corresponding tasks. These n-ary relationships are stored in the ABox, using concepts from the TBox.
- Instructional texts and references to image and video materials. Each potential task is assigned a text (in English and German), a reference to an image, and a reference to a video where the execution is demonstrated. Like DIY concepts, instructions are arranged in a hierarchy (in the TBox).

This information provides a common basis both for the plans generated by the planning component and the information passed on to the user through dialogue management and user interface. Reasoning is used to establish all facts that are implicit in the formalisation (e.g. that an instance of a particular type of drill driver is also a drill).

1 $\mathit{PSR18Li2} \sqsubseteq \mathit{DrillDriver}$
2 $\mathit{DrillDriver} \sqsubseteq \mathit{Drill}$
3 $\mathit{PhilipsScrewBit} \sqsubseteq \mathit{ScrewBit}$
4 $\mathit{PhilipsScrewBit} \sqsubseteq \exists \mathit{hasVisualFeature.CrossShape}$
5 $\mathit{PhilipsScrew} \sqsubseteq \mathit{Screw}$
6 $\mathit{WoodScrew4mm} \equiv \mathit{WoodScrew}$
    $\sqcap \exists \mathit{hasDiameterInMM} = \text{“4”}$
7 $\mathit{WoodScrew4mm} \sqsubseteq \mathit{WoodScrewBetween4And6mm}$

(a) TBox excerpt.

1 $\mathit{PSR18Li2(drill-1)}$
2 $\mathit{Softwood(back)}$
3 $\mathit{Softwood(tray)}$
4 $\mathit{HoleShape(round-hole-3mm)}$
5 $\mathit{PhilipsWoodScrew(screw-1)}$
6 $\mathit{hasDiameterInMM(screw-1, \text{“3”})}$

(b) ABox excerpt.

Fig. 4. Excerpt of ontological facts in the TBox (a) and the ABox (b) for the running example.

1 $\mathit{ScrewConnectionConfig(config1)}$
2 $(\exists \mathit{ScrewConnectionConfig\_materialType1.Softwood})$
3 $\quad \mathit{(config1)}$
4 $(\exists \mathit{ScrewConnectionConfig\_materialType2.Softwood})$
5 $\quad \mathit{(config1)}$
6 $(\exists \mathit{ScrewConnectionConfig\_screwType.}$
$\quad \mathit{WoodScrewBetween4And6mm)(config1)}$
7 $(\exists \mathit{ScrewConnectionConfig\_holeShape1.RoundHole3mm})$
8 $\quad \mathit{(config1)}$
9 $\mathit{ScrewConnectionConfig\_rotarySpeed(config1,1800)}$

Fig. 5. Example for a configuration stored in the ontology.

6.1. Factual knowledge for planning

The generation of a suitable plan depends on available tools and materials and their properties. A description of what objects are available and their general properties is passed on to the planner to become part of the initial state of the planning problem. For example, if the ontology specifies in its ABox that an instance of a *DrillDriver* is available, all inferable properties with respect to the TBox (e.g. that a *DrillDriver* can be used as a *Drill*, and therefore also this particular machine) are also available to the planner. *Concepts* in the ontology correspond to *types* in the planning domain and individual objects are modelled as constants in the planning domain. For example, the assertion *DrillDriver(drill-1)* is represented as `drill-1 - DrillDriver` in PDDL syntax, which declares that the object `drill-1` exists and is a member of the type `DrillDriver`. When requested, the ontology manager transfers a list of all known instances in this format to the planner.

The information modelled in the ontology includes n-ary relationships in the ABox that represent valid combinations of tools, materials and settings (henceforth referred to as “configurations” [10]). For instance, one might stipulate that a valid combination of parameters for connecting two pieces of softwood using a wood screw of 4 to 6 mm diameter involves predrilling a hole of 3 mm diameter with moderate rotary speed. In first-order logic, one could, for example, write:

$$\begin{aligned}&\forall x; y; z; q : \mathit{Softwood}(x) \land \mathit{Softwood}(y)\\ &\land \mathit{WoodScrewBetween4And6mm}(z)\\ &\land \mathit{RoundHole3mm}(q)\\ &\rightarrow \mathit{screwConnectionConfig}(x, y, z, q, 1800)\end{aligned}$$

This statement refers to a so-called concept product (cf. [36]); a property is defined (in this case *screwConnectionConfig*) that connects together the instances of several concepts (*Softwood*, *WoodScrewBetween4And6mm*, *RoundHole3mm*), thus corresponding to the Cartesian product of the concepts' set-theoretic interpretation. However, concept products are not natively supported by typical description logics and ontology reasoners. Since ontology languages are usually restricted to binary relations (properties), we use reification to express such n-ary relationships. For each combination of parameters, we introduce an individual to represent the combination and link it to its elements using several binary (concrete or abstract) properties. As an example, consider the set of ABox assertions in Fig. 5. The individual *config1* is introduced to link together several attributes (material type, screw type, rotary speed), where attributes are either specified using the form $(\exists r.A)(c)$, where c is the introduced individual and r the property representing the attribute, or they are specified using data properties (to assign data values such as the number 1800 in the example). These specifications are passed on to the planner in PDDL format, in case of the example:

```
config1 - ScrewConnectionConfig
config1 - Config
(ScrewConnectionConfig_materialType1
     config1 Softwood)
```

```
(ScrewConnectionConfig_materialType2
     config1 Softwood)
(ScrewConnectionConfig_screwType
     config1 WoodScrewBetween4And6mm)
(ScrewConnectionConfig_holeShape1
     config1 RoundHole3mm)
(ScrewConnectionConfig_rotarySpeed
     config1 1800)
```

The first two assertions declare that the object `config1` is a member of both the `ScrewConnectionConfig` and the `Config` types. Note that `config1 - Config` is obtained by inference (since the ontology entails *Config*(*config1*)). The latter five assertions declare facts that are true in the initial state. Normally, the arguments of such facts must be objects (while numbers like 1800 are treated as objects) and not types, as is the case when configurations are translated. How the planner handles these kinds of facts is described in Sec. 7. In planning, these configurations serve as a kind of forall-statement (as motivated above). That is, these configurations are used to specify which combinations of parameters are allowed for instantiating a task (for instance, all instances of *WoodScrewBetween4And6mm* are candidates according to the example configuration), as illustrated in Sec. 7.

6.2. Instructions

Whereas planning ensures that the proposed actions are executable and lead to the goal, the concrete instructions provided to users of an assistance system need to convey how a task is actually carried out. This is particularly true if Alice is about to perform an elementary operation for the first time. For each task, we provide a text in English/German (authored by following manuals and textbooks), an image showing the operation being carried out, and a video. These collections of texts and links to the associated media are maintained in the ontology. Planning instantiates the generally available tasks with parameters, for instance, when a particular drill is selected for instantiating a drilling task. For any combination of parameters (e.g. a drill driver *PSR18Li2* together with a 15 mm Torx screw, to pick up the example from Sec. 4), the best fitting instructional material needs to be retrieved from the ontology (e.g. an image showing the specific type of drill with a specific attachment, if possible). For this, we use classification: A taxonomy represents classes of actions, which are characterised by properties such as “uses a drill of type A”.

When a plan is generated, its tasks and actions are instantiated with parameters. For instance, assume that planning instantiates the *Connect_Screw* method with the following arguments (specifying that drill-1 and screw-1 are to be used to connect two objects):

```
ConnectScrew(drill-1, object-1, object-2,
screw-1)
```

Using classification in the ontology, the most specific instruction is determined that fits the description of an instantiated task, and the relevant materials are provided to be shown in the user interface. To enable classification, an instance *i* representing this task is created in the ABox together with its (reified) arguments (where the auxiliary properties *instr_arg1–n* connect the task with its arguments):

instr_name(i,Connect_Screw)
instr_arg1(i,drill-1)
instr_arg2(i,object-1)
instr_arg3(i,object-2)
instr_arg4(i,screw-1)

In the ontology’s TBox, concepts define classes of tasks and actions. Subsumption is used to define a taxonomy of (increasingly specific) instructions. For instance, the axiom

$$\mathit{Instr\_Connect\_Screw\_PSR18Li2} \equiv \mathit{Instr\_Connect\_Screw} \sqcap \exists\, \mathit{instr\_arg1.PSR18Li2}$$

defines that the concept *Instr_Connect_Screw_PSR18Li2* is a refinement of *Instr_Connect_Screw*. Further axioms specify the associated properties and data (references to image and video content) for the defined instruction concepts.

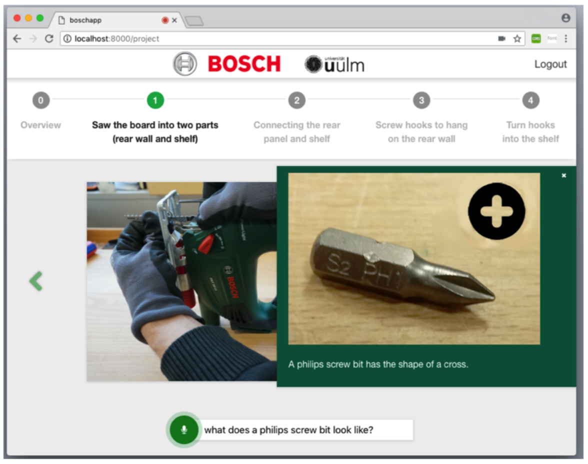
Classification establishes that:

Instr_Connect_Screw_PSR18Li2(i),
Instr_Connect_Screw(i),
$\mathit{Instr\_Connect\_Screw\_PSR18Li2} \sqsubseteq \mathit{Instr\_Connect\_Screw},$

so *Instr_Connect_Screw_PSR18Li2* is the most specific *Instruction* concept that fits the task under consideration. The associated video, image and text are retrieved for presentation.

6.3. Generated textual descriptions

As briefly mentioned in Sec. 4, the user may query the system for the available information pertaining


boschapp
localhost:8000/project
BOSCH
uulm
Logout
0
Overview
1
Saw the board into two parts (rear wall and shelf)
2
Connecting the rear panel and shelf
3
Screw hooks to hang on the rear wall
4
Turn hooks into the shelf

A philips screw bit has the shape of a cross.

what does a philips screw bit look like?

Fig. 6. The system's reply to "What does a Philips screw bit look like?".

to a particular object or category of objects. For instance, the user might ask "What is a PSR18Li2?" or "What does a Philips screw bit look like?". A textual description is then generated using the information formalised in the ontology. For a general description ("What is a PSR18Li2?"), the ontology axioms relating to the concept (or to an instance of a concept) are collected, including inferred axioms. The axioms are ordered heuristically; first simple subsumptions (e.g. *PSR18Li2*⊑*DrillDriver*; "A PSR18Li2 is a drill driver") are output, then axioms specifying the use of the object (identified by a property *isSuitedFor*), and lastly the remaining properties. For the generation of text from axioms, we use the verbalisation tool presented by Schiller et al. [27]. The visual appearance associated with a concept is formulated using a dedicated property *hasVisualFeature*, such that these axioms are retrieved for questions such as "What does a Philips screw bit look like?", yielding, for example: "A Philips screw bit has the shape of a cross". These answers are further supplied with an image, as shown in Fig. 6.

6.4. Generated explanations

As briefly mentioned in the introduction, facts inferred from the ontology can be justified by the facts from which they were derived and the inference steps by which they were obtained. To take up the example from the introduction, consider the statements in the ontology shown in Table 1. Together these facts entail the statement $\mathit{Screwing3MMScrewIntoSpruce} \sqsubseteq \exists\, \mathit{doesNotRequire}.\mathit{Predrilling}$. We use a rule-based reasoner together with a template-based mechanism to translate the generated proofs to natural language text (cf. [27]). We exemplify this process with the first statement of the generated text presented in the introduction:

Spruce is softwood, thus screwing a 3 mm screw into spruce applies to softwood.

The statement *"screwing a 3 mm screw into spruce applies to softwood"* is a logical consequence of the first and the last axiom in Table 1. This is the result of the application of two inference rules. First the conjunction of the equivalence (the last axiom in Table 1) is eliminated, yielding:

$$\mathit{Screwing3MMScrewIntoSpruce} \sqsubseteq \exists\, \mathit{materialType}.\ \mathit{Spruce}$$

Then a second inference step applies:

$$\frac{\begin{array}{c}\mathit{Screwing3MMScrewIntoSpruce} \sqsubseteq \exists \mathit{materialType}.\ \mathit{Spruce} \\ \mathit{Spruce} \sqsubseteq \mathit{Softwood}\end{array}}{\mathit{Screwing3MMScrewIntoSpruce} \sqsubseteq \exists \mathit{materialType}.\mathit{Softwood}}$$

A straightforward approach to verbalisation would now for every step first turn all of the premises into text, and then the conclusion. This results in long, repetitive output. To make the texts shorter, we employ some heuristics (which are discussed in more detail by Schiller et al. [27]). Firstly, we distinguish between those inference rules that are to be verbalised, and a set of inference rules that are considered too trivial for verbalisation. This applies, for instance, to conjunction elimination as employed above. So the intermediate conclusion that *"screwing a 3 mm screw into spruce applies to spruce"* is not output. Secondly, we keep track of those statements that we assume Alice to be already aware of, including those statements that are implied by unverbalised inference rule applications as above. So when the second inference rule is applied, only one of the premises (*"spruce is softwood"*) is being output together with the conclusion. A more detailed discussion of the employed techniques aimed at shortening the generated explanations is provided by Schiller et al. [27].

7. Planning

The instructions presented to Alice by ROBERT are based on an automatically generated plan. This ensures that the instructions achieve the user's objective. It also

Table 1

Ontological facts from which $Screwing3MMScrewIntoSpruce \sqsubseteq \exists doesNotRequire.Predrilling$ can be derived. Note that labels in the ontology are used to replace concept and property names in the verbalisation with names that are deemed more natural to users, e.g. "applies to" instead of "materialType".

Axiom	Verbalisation
$Spruce \sqsubseteq Softwood$	"Spruce is softwood."
$Screw3MM \equiv Screw \sqcap \exists\, hasDiameterInMM = \text{“3”}$	"A 3 mm screw is a screw that has a diameter of 3 mm."
$Screw \sqcap \exists hasDiameterInMM = \text{“3”} \sqsubseteq SmallScrew$	"A screw that has a diameter of 3 mm is a small screw."
$ScrewingSmallScrewIntoSoftwood \equiv \exists\, materialType. Softwood \sqcap \exists\, screwType.\, SmallScrew$	"Screwing a 3 mm screw into softwood is something that applies to softwood and that is done with a small screw."
$ScrewingSmallScrewIntoSoftwood \sqsubseteq \exists\, doesNotRequire. Predrilling$	"Screwing a small screw into softwood does not require predrilling."
$Screwing3MMScrewIntoSpruce \sqsubseteq \exists\, materialType. Spruce \sqcap \exists\, screwType.\, Screw3MM$	"Screwing a 3mm screw into spruce is something that applies to spruce and that is done with a 3 mm screw."

provides a suitable mechanism for adapting ROBERT's assistance to different environments, e.g. to the tools and materials available to the user. Without planning, we would have to manually specify instructions for every imaginable situation, which is, due to the variability in the DIY setting, simply impossible in practice. Planning allows ROBERT to find instructions even in previously unknown situations, simply by providing the planner with a description of the current state, and thus to find best-fitting instructions for the given circumstances. This is especially important as the available tools and materials are expected to differ widely from user to user and situation to situation. Before the assistant has been started by Alice for building a key rack, ROBERT does not know the type of wood Alice has (e.g. hard or soft, which kind of tree) or the sizes and types of screws, if she even has screws – she might just have a set of nails. ROBERT is only informed about these circumstances once it is activated by the user and thus has to flexibly adapt to the new situation without the possibility of prior knowledge. ROBERT can deal with a wide variety of tools, each differing in their capabilities and possible configurations. For example, a Bosch IXO does not have a gear switch, while a PSR18Li2 does. If ROBERT is to assist Alice in using the latter drill, it should instruct her to configure the gear switch according to the material she is drilling into, but not do so if she has a Bosch IXO. ROBERT's planner adapts the given instructions to these conditions at run-time, based on the procedural planning model and the information about these devices stored in ROBERT's ontology. This allows ROBERT to provide instructions that are fitting exactly to the situation at hand – in contrast to, e.g. a generic instructional video.

ROBERT's lifted model-driven approach allows for easy scalability to both new tools and materials as well as to new types of projects. For example, whenever a new model of electric drills is released, it suffices to add a description of its abilities and configuration options to the ontology. Based on our coupling between planning model and ontology, this information will automatically be taken into account by the planner and instantly enable instructions using this new device to the best of its capabilities.

We start by introducing the primitive action theory used by ROBERT and describe its connection with ROBERT's ontology. Thereafter, we show how ROBERT uses hierarchical planning to formulate its goals and detail on the advantages we draw from ROBERT's hierarchical planning model.

7.1. Action model and connection to the ontology

ROBERT's planning domain contains lifted descriptions of a wide variety of possible actions in the DIY environment. This includes (but is not limited to): adding and removing batteries, adding/removing drill bits and saw blades, drilling holes, putting screws in them, inserting pegs into holds, sanding surfaces, fixing objects together, etc. These lifted descriptions provide a factorised representation of a multitude of actions with structurally similar preconditions and effects. For example, there is only one action to attach a battery to a device in the model. Its action description defines two parameters – the battery and the device. Every assignment of these two parameters to objects in the model constitutes a valid instantiation of the action for attaching a battery. Based on the lifted model and all available objects, we can compute all valid instantiations (called groundings) of all actions.

The actions in ROBERT's planning model are specified in a fairly general fashion. This generality al-

```
(:action Drill_Screw
  :parameters (?drill - Drill
          ?o1 - Connectable ?o2 - Connectable
                  ?screw - Screw)
  :precondition (and
   (usable ?screw) (usable ?o1) (usable ?o2)
   (imply (typeOf ?o1 HomObj) (fixated ?o1))
   (imply (typeOf ?o2 HomObj) (fixated ?o2))
   (exists (?b - Battery) (and (AttachedBattery ?drill ?b) (hasEnergy ?b)))

   (exists (
    ?sb - ScrewBit
    ?sbh - ScrewBitHolder
    ?screwType ?screwBitType - Type
    ?rpm - Number
    ?ds - DrillSettings
   ) (and
    (AttachedShank ?drill ?sbh) (AttachedShank ?sbh ?sb)
    (typeOf ?sb ?screwBitType)
    (typeOf ?screw ?screwType)
    (Drill_settings ?drill ?ds)
    (DrillSettings_direction ?ds right)
    (DrillSettings_rotarySpeed ?ds ?rpm)
    (exists (?sc - ScrewingConfig) (and
      (ScrewingConfig_screwType ?sc ?screwType)
      (ScrewingConfig_screwBitType ?sc ?screwBitType)
    ))
    (exists (?scc - ScrewConnectionConfig ?ot1 ?ot2 - Type) (and
      (typeOf ?o1 ?ot1)
      (typeOf ?o2 ?ot2)

      (ScrewConnectionConfig_screwType ?scc ?screwType2)
      (ScrewConnectionConfig_materialType1 ?scc ?ot1)
      (ScrewConnectionConfig_materialType2 ?scc ?ot2)
      (ScrewConnectionConfig_rotarySpeed ?scc ?rpm)

      (forall (?hs1 - HoleShape) (and
        (imply (ScrewConnectionConfig_holeShape1 ?scc ?hs1) (holeShape ?o1 ?hs1))
      ))
      (forall (?hs2 - HoleShape) (and
        (imply (ScrewConnectionConfig_holeShape2 ?scc ?hs2) (holeShape ?o2 ?hs2))
      ))
    ))
   ))
  )
  :effect (and
   (not (usable ?screw))
   (connected ?o1 ?o2 ?screw)
  )
)
```

Fig. 7. Declaration of the action `Drill_Screw` in ROBERT's planning model.

lows for automatic adaptation to, e.g, new tools, without the need to alter the planning model. For example, the action `Drill_Screw` represents driving a screw `?screw`[5] through two objects `?o1` and `?o2` using any kind of electric drill `?drill` in order to connect them with each other. Each of the involved objects is represented by a parameter of the action `Drill_Screw`. Fig. 7 contains a simplified version of the action that is contained in our model. ROBERT's model contains further preconditions that pertain to further configuration options of drills. Clearly, it is not possible in the real world to execute `Drill_Screw` with an arbitrary combination of materials, drills, and screws, i.e. an arbitrary combination of parameters. For example, one cannot use a Bosch IXO drill and a wood screw to connect a metal plate to a concrete wall. `Drill_Screw`'s preconditions first check the *procedural* requirements of driving a screw: that the screw and the objects we want to connect are usable (e.g. the screw is not already connected to something else), and fixated if necessary, and that a charged battery is connected with the drill. This information is purely procedural and as such fully modelled in the planning domain.

The second part of `Drill_Screw`'s preconditions

[5]Names of variables in PDDL start with a question mark.

asserts that the configuration in which we are using the drill is actually allowed. These rules are not modelled inside the action's preconditions, but are stored in ROBERT's ontology, since they constitute factual knowledge. `Drill_Screw` references this information and checks whether the state we are currently in complies with it. First, `Drill_Screw` checks whether the bit inserted into the drill matches the screw we want to use (line 24-27). Second, `Drill_Screw` checks whether the drill's settings (e.g. rotation speed) are compatible with the material we are driving the screw into and that connecting the objects `?o1` and `?o2` using the selected screw and drill is possible (line 28-43). Both of these checks use our *configuration* mechanism [10], which is based on storing allowed configurations inside the ontology and referencing them in the planning model. As described in Sec. 6.1, configurations are stored in the ontology as reified n-ary connections between concepts, numbers, truth values, and individuals. An example can be found in Fig. 5. Each individual axiom of such a configuration is transferred to the planner in form of a triple `(r i X)`, where `r` is the role, i.e. a property of the configuration and `i` is the instance representing the configuration. `X` can either be an individual in the ontology, a number, a truth-value, or a concept. Individuals correspond to objects in the planning model and we similarly interpret numbers and truth values as ordinary objects. These triples express that a specific object `X` is part of the configuration.

In case `X` is a concept `C`, the situation is slightly more complicated. Consider a given configuration instance $\mathtt{i}_{\mathtt{conf}}$ with corresponding triples $(\mathtt{r}_1\ \mathtt{i}_{\mathtt{conf}}\ \mathtt{C}_1), \ldots, (\mathtt{r}_n\ \mathtt{i}_{\mathtt{conf}}\ \mathtt{C}_n)$. They denote a valid configuration for any combination of objects $\mathtt{o}_1 \in \mathtt{C}_1, \ldots, \mathtt{o}_n \in \mathtt{C}_n$. Such a configuration thus specifies a restriction based on sets of objects (which correspond to individuals) and not on individual objects. In the planning model, we could interpret any such triple as a fact `(r i C)`. Unfortunately, PDDL lacks support for such expressions, as the arguments of facts are restricted to be objects and not types (which mirror concepts in the ontology). We have therefore extended the allowed expressions of PDDL to also allow for such expressions and we introduced a `typeOf` keyword, which determines for an object `?o` its type `?ot` [10].[6] This way, the configuration shown in Fig. 5 as stored in the ontology is transformed into five facts contained in the initial state of the planning problem, shown in Fig. 8. This enables us to refer to these configurations in `Drill_Screw`'s preconditions. There we check whether a configuration exists that allows for executing `Drill_Screw` in the state in which we are currently in. As such, we qualify in line 28 of Fig. 7 the reified configuration `?scc`, which allows driving a screw in the current state. Since the configuration relates concepts to each other, we first have to determine the types of all relevant objects (see lines 19–20 and 29–30). Based on these types, we check whether the selected configuration allows for relating constants of these types (lines 32-35). Further, there are cases where configurations are not necessarily complete. For example, the configuration shown in Fig. 8 specifies a `holeShape1` but not a `holeShape2`, indicating that the top object through which the screw is driven must be pre-drilled, but not the bottom object. This is expressed in the `Drill_Screw` action in lines 37–42.

We use the same pattern of checking configurations in all primitive actions inside the planning model. As such, the planner requests, at the beginning of each planning process, all known configurations from the ontology and adds all returned facts to the initial state. This modularity is central for the scalability of ROBERT: whenever a new tool is added, it suffices to add new configurations for it to the ontology (if this is even necessary, as configurations can also refer to high-level concepts). The planner will automatically load them from the ontology and will be able to apply, e.g. `Drill_Screw` with a newly added tool and will be able to configure it appropriately.

We also use ROBERT's primitive action model to ensure that we never provide instructions to the user that might lead her to a dangerous situation. For example, some devices are in general considered dangerous if a battery is connected to them – most notably this holds for electric saws. For those devices one should not change attachments if the battery is still attached, which we can check in the action that adds and removes sawing blades for the saw and other similarly dangerous devices. Further we can use preconditions to, e.g. check that Alice wears gloves and safety glasses when performing potentially dangerous actions.

7.2. Abstraction hierarchy and HTN planning

As stated in Sec. 2, ROBERT uses HTN planning to formalise its planning domain. HTN planning domains

[6]Technically, we introduce a new object `s_C` for each type `C` and thereafter treat `s_C` as an ordinary object, while adding all necessary instances of the `typeOf` predicate to the initial state. The `typeOf` predicate can be handled as an ordinary predicate on the constants `s_C` representing types.

```
(ScrewConnectionConfig_materialType1 config1 Softwood)
(ScrewConnectionConfig_materialType2 config1 Softwood)
(ScrewConnectionConfig_rotarySpeed config1 1800)
(ScrewConnectionConfig_screwType config1 WoodScrewBetween4And6mm)
(ScrewConnectionConfig_holeShape1 config1 RoundHole3mm)
```

Fig. 8. Example for a configuration converted into facts for the initial state.

distinguish two types of tasks: primitive actions and abstract tasks. `Drill_Screw` is an example for an action in ROBERT's domain. From the planner's point of view, there is no need to refine `Drill_Screw` into more basic steps as all (potential) variability in executing it can be handled by the dialogue manager. Abstract tasks $t \in T$, on the other hand, model more complex, high-level activities. Consider as an example `Connect`, which represents the act of connecting two objects. It is abstract as (a) it can consist of several primitive actions, e.g. fixating the two objects, (potentially) pre-drilling holes for the screw, configuring the drill for screwing, driving the screw, and loosening the fixation again, and as (b) there is multitude of options to conduct `Connect`, e.g. by using screws, nails, or pegs.

Fig. 9 specifies the method `Connect_Screw` as an example. It is expressed using an extension of PDDL for expressing HTN domains. The method `Connect_Screw` provides a means to decompose the abstract task `(Connect ?o1 ?o2)` (line 3). The high-level semantics of `Connect` is that if executed, it ensures that its two arguments `?o1` and `?o2` are connected. This abstract task however does not specify how this connection is to be made nor does it specify any tools used to connect `?o1` and `?o2`. The method `Connect_Screw` specifies one possible means to achieve the goal represented by `Connect`: driving a screw through both objects. To this end, it specifies a sequence of (both primitive and abstract) tasks (lines 5-11), that, if executed, will establish the connection. Here, each line specifies an individual subtask of the method, i.e. a task in the method's task network. The method also specifies the order of the task network in saying that it is a total order of the subtasks (line 4). First, holes are (potentially) made into `?o1` and `?o2` (lines 5 and 6), a process called pre-drilling. These tasks are again abstract and offer two options for decomposition: either drill a hole or do nothing. This generality is necessary as there are materials and screws where pre-drilling none, one, or both objects is necessary. The determination which of the two objects is to be pre-drilled is made by the planner by choice of suitable decompositions. Here, configurations again come into play. Connecting `?o1` and `?o2` will only be possible if both objects are appropriately pre-drilled. The method itself does not take this information into account with the aim of keeping the planning domain as general as possible. This generality again enables us to integrate new tools and materials without the need to alter the planning model at all.

After these two abstract tasks are completed, the method `Connect_Screw` ensures that a battery and some shank object (e.g. a screw bit) are inserted into the drill `?drill`, i.e. that the drill is properly prepared. Again these two tasks might do nothing, as, e.g. a battery may already be present in the drill. Similarly, the configuration checked in `Drill_Screw` ensures that the correct bit has been inserted into the drill. Next, the two objects to be connected may have to be fixated, then the screw is driven using the selected drill `?drill`. Lastly, the two objects can be un-fixated. After these tasks have been completed, `?o1` and `?o2` are connected.

The goal formula used in ROBERT enforces that the work area is cleaned up after the project and that all devices are stored properly. The initial abstract plan specifies the project itself, i.e. it consists of abstract tasks that, if executed, ensure that the project is completed. Since we cannot expect users to specify these plans, we provide a library of possible initial plans, i.e. DIY projects. Users can simply select the project they want to build. Afterwards, the planner generates a plan for this project, which adapts the project to the users' individual circumstances.

To generate the plan based on the domain, the initial state, and the goal description, we use the HTN planning system PANDA [37]. It offers a wide variety of algorithms for solving HTN planning problems. In ROBERT we use a configuration which translates the HTN planning problem into a propositional formula and then passes it on to a SAT solver [31, 38, 39]. From the resulting valuation, we can extract both the primitive actions necessary to complete the project, as well as the decomposition hierarchy that led to this particular plan.

```
(:method Connect_Screw
 :parameters (?drill - Drill ?o1 - Connectable ?o2 - Connectable ?screw - Screw)
 :task (Connect ?o1 ?o2)
 :ordered-subtasks (and
   (MaybeMakeHole ?o1)
   (MaybeMakeHole ?o2)
   (EnsureBattery ?drill)
   (EnsureShank ?drill)
   (MaybeFixateTwo ?o1 ?o2)
   (Drill_Screw ?drill ?o1 ?o2 ?screw)
   (MaybeUnFixateTwo ?o1 ?o2)
 )
)
```

Fig. 9. Example of a method for connecting two objects `?o1` and `?o2` using a single screw `?screw`.

7.3. Exploiting the task hierarchy

In addition to flexibility when it comes to the definition of the goal, HTN planning also allows us to significantly enhance ROBERT's abilities to communicate the contents of the individualised project to the user. We use the decomposition hierarchy returned by the planner to create an abstract version of the found primitive plan tn_{Sol}. Technically, this plan is a task network tn^* containing abstract tasks that can be reached from the initial plan via decomposition and from which the solution tn_{Sol} can be reached via decomposition. In essence, this task network is an intermediate refinement step on the way to the solution tn_{Sol}. We compute tn^* by applying the decomposition methods used to obtain tn_{Sol} from tn_I backwards, i.e. by replacing the tasks contained in a method by the method's abstract task. This process starts with the solution tn_{Sol} and is repeated until a task network of the desired level of abstraction has been reached. We further apply methods such that we achieve uniform decomposition depth for all tasks in tn^*. The depth of a task t is the number of consecutively applied decompositions needed to reach t from the initial plan. This depth corresponds to the level of abstraction a task has. For backwards application of methods, we always consider tasks with the highest decomposition depth. This ensures that the tasks in the determined task network tn^* are roughly at the same level of abstraction.

At the moment, we create only one abstract plan tn^* by reducing the size of the task network until it has less than seven tasks (cf. capacity of the human short-term memory [40]). In our key-rack example, this abstract plan consists of four abstract tasks (see Fig. 10). These correspond to the four basic tasks in building a key rack:

(1) sawing a plank into two boards,
(2) connecting the boards,
(3) attaching two hangers to the back, and
(4) adding four hooks to the tray.

```
Cut(board,back,tray)
Connect(back,tray)
Attach(hanger,2)
Attach(hook,4)
```

Fig. 10. Abstract plan found in the key-rack example.

ROBERT uses the abstract plan tn^* in two ways.

First, ROBERT shows Alice a progress bar for her project based on tn^* at the top of the user interface (see, e.g. Fig. 2). Every primitive action in the solution tn_{Sol} is derived via decomposition from exactly one task t in tn^*. Whenever we present a primitive action to the user, we highlight the abstract task t in the progress bar. This display serves both as an orientation to the user, as well as provides instant positive feedback. Thus, Alice can keep track of her progress in completing the project. The advantage of using an abstraction for this purpose is that the completion of a step shown in the progress bar corresponds to completing a logical activity within Alice's current project.

In addition to showing the user an automatically generated progress bar, we use the abstraction to determine the numbering of instructions. Instead of globally numbering steps ("step 12 of 42"), we assign a chapter-like numbering to primitive actions ("2.5 of 2.10", i.e. 5 of 10 in part 2). This leads to a smaller number of steps-to-be-performed shown to the user, which again serve as a motivational factor.

Second, we are able to provide Alice not only with instructions based on the detailed sequence of primitive actions, but also on a more abstract level. As such, we present the steps of tn^* in the same way as we present primitive actions. This is especially useful for the user to get an overview of the project and to only then perform the detailed instructions of the plan. It is also more suitable for users who might be able to perform individual steps on their own using their prior knowledge, but lack the required skills for other tasks. Alice can choose on a task-per-task basis whether abstract instructions suffice or whether more detailed instruction is necessary. See the following Sec. 8 for further details.

8. Dialogue management

Traditionally, the task of a dialogue manager of a user interface is to receive a semantic representation of the user input, spoken or expressed through other modalities, and, depending on the application as well as on the dialogue state, to select an appropriate system reaction. In ROBERT, the dialogue management controls the communication between the user and the components providing assistance, i.e. ontology manager and planner. Furthermore, it is responsible for the system's robustness by compensating for low speech recognition performance (e.g. confidence measures) and by establishing common ground with the user (confirmation strategies). A detailed description of the implementation and functionality of our dialogue manager is provided in the following. We provide insight into the concepts of our approach before we focus on the implementation.

8.1. Concepts

For rendering our system ubiquitously accessible and to allow for a modular extensible architecture, we follow a distributed client/server-based approach. The client is formed by a browser interface, which is able to process multimodal user input and to present multimedia content. On the server side a HTTP web service handles requests from the interface and conveys the user input to the dialogue component. The web service is designed following the REST (REpresentational State Transfer) paradigm [41]. The main advantages of this paradigm are architectural constraints intended to reduce latency and network communication, while optimising the independence and scalability of server components. However, our web service is not RESTful to a full extent, as we cannot render the server stateless due to the fact that tracking of dialogue state and history are essential for dialogue management development.

Another main concept of our approach is to make use of cloud-based cognitive services for automatic speech recognition, language understanding, and text-to-speech synthesis. Relocating these services into the “cloud” holds several advantages, but also some disadvantages, which we discuss as well. As these operations require considerable processing power, a major advantage lies in the saving of computing capacity allowing for a more efficient implementation of the user interface and lower latency.

Furthermore, cloud-services guarantee a high degree of modularity. As a vast number of individual services can easily be added by using endpoint queries, the overall architecture is extensible, flexible, and offers great maintainability.

A typical disadvantage of cloud-based approaches is the need for a steady internet connection in order to function properly. Even nowadays, there exist situations (e.g. a user is working in the basement) in which internet connectivity cannot be guaranteed. In such cases restrictions have to be made regarding a natural language interaction, as speech and intent recognition requires server connection. Therefore, the system needs to be adaptive and provide a "graceful degradation", i.e. maintain basic system functionality, even in case of a missing internet connection. This topic is currently work in progress, and hence not handled in this paper.

For the development of our dialogue manager , we chose an agent-based approach. Characteristically for this approach, the dialogue is controlled by multiple agents that are capable of reasoning about their actions [42, 43]. Moreover, the dialogue is not statically predefined and evolves dynamically, whereby the dialogue flow is determined at runtime dependent on the current world state and the agent's context. This makes the approach especially useful in dynamically changing application domains, e.g. assisting a user in the execution of a complex task. As each agent can be modelled individually, the benefits from several different dialogue control models and management strategies can be combined. Our approach utilises agents for interacting with the planning and knowledge module as well as an agent handling meta-dialogues, such as information grounding. The implementation of the concepts is described in the following section.

8.2. Implementation

The multimodal dialogue system of ROBERT consists of a user client interface and of server-based dialogue agents [44]. While the interface receives user input and presents system output, the dialogue agents are responsible for deciding on appropriate system actions, e.g. to provide explanations on according user request.

8.2.1. User client interface

As front-end to our system a multimodal graphical interface is implemented using the *Vue.js*[7] framework. The JavaScript-based framework allows for creating incrementally scalable web pages, which are easy to maintain and offer a high reactivity using dynamic data binding. Those features are deemed best for developing a sophisticated multimodal web interface which provides methods for handling dialogue input as well as output. Our interface supports three different input modalities: Speech, text, and touch/click input.

Due to the need for hands-free communication during the execution of a DIY project, speech input serves as the primary modality. We use the Google Chrome web browser's own speech-to-text service in combination with the *annyang.js*[8] JavaScript library. The *annyang.js* library provides a keyword-spotting ability, enabling ROBERT to separate between requests of Alice directed at the user interface and those involving the reasoning capabilities of ROBERT.

When Alice solely wants to control elements of the user interface, e.g. navigate through the presented slides, an interaction with the backend of ROBERT is not required. As such, simple commands like 'next' or 'go back' are spotted by *annyang.js* and processed without informing the dialogue manager.

For addressing the cognitive modules of ROBERT a certain keyword is necessary. After saying that keyword, subsequent speech input is converted into text and forwarded to the system for further processing. The purpose of this approach is to avoid that ROBERT reacts to off-topic talk as, per default, speech recognition is permanently activated, unless the user disables speech input by pushing a button. Instead of speech input, it is also possible to communicate directly to the system via text input and to navigate through the slides showing plan steps by touching direction arrows or swiping gestures on the tablet's screen.

To provide visual queues to Alice, whenever an instructional text is displayed on the user interface, we automatically highlight those words corresponding to concepts in the ontology. This tells Alice for which words more detailed descriptions and explanations are available (see Fig. 11). Moreover, these highlighted words can also be used as links, i.e. Alice can request an explanation for them via touch.

[7]https://vuejs.org/

[8]https://www.talater.com/annyang/

Dialogue output is presented depending on the user's request. If Alice has requested ROBERT to assist her in a DIY project, i.e. has uttered a request for planning, the resulting plan is shown as a sequence of slides using the *vue-slick.js* library, where each individual slide corresponds to one plan step of the plan π found by the planner. If Alice has requested an explanation or background information, the respective content is conveyed in the form of text and synthesised speech, also using Chrome's speech service. Fig. 11 shows a slide instructing the user to perform an action (here inserting the saw blade) that has been generated based on the plan step passed on by the planner.

8.2.2. Server-based dialogue agent

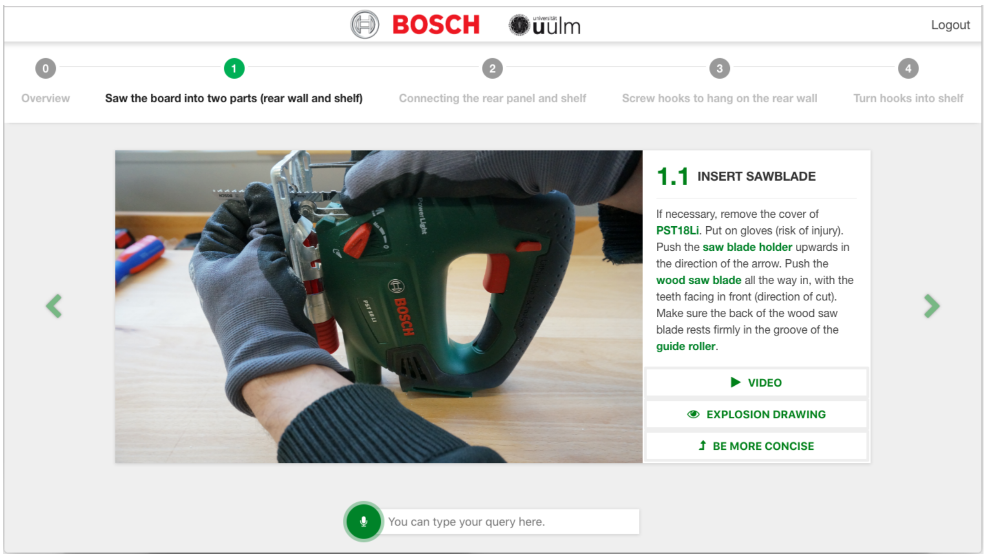
User input from the client interface is sent to an HTTP API server used as the platform for dialogue management. The service can be accessed by using standard HTTP methods. For manipulating the resource, we make use of create, read, update, and delete (CRUD) messages:

create is used to add new users via the HTTP POST method. New users have to register for the service by setting up a user profile including login data for later authentication. Upon registration, a new user is assigned to the resource /users, creating a subordinate instance /users/{id}. For handling multiple, parallel user dialogues, each user instance obtains its individual dialogue manager and can be accessed by sending requests to the resource /users/{id}.

read is used for authentication and retrieving individual user profiles. In order to use ROBERT, a registration and login is required. Via the HTTP GET method, authentication information from all users is gathered from the resource /users and used for individual user validation. Furthermore, the method retrieves a specific user profile when called on an ID resource. This can be applied, for example, for individualisation of the user interface depending on the current user profile.

update is used to forward user input to the dialogue manager via the HTTP PUT method on the resource /users/{id}. The arriving input is processed by the individual dialogue manager instance of the respective user and the result passed back to the user interface.

delete is used to delete an individual user profile or all user information at once via the HTTP DELETE method.


BOSCH
uulm
Logout
0 Overview
1 Saw the board into two parts (rear wall and shelf)
2 Connecting the rear panel and shelf
3 Screw hooks to hang on the rear wall
4 Turn hooks into shelf
1.1 INSERT SAWBLADE
VIDEO
EXPLOSION DRAWING
BE MORE CONCISE
You can type your query here.

If necessary, remove the cover of **PST18Li**. Put on gloves (risk of injury). Push the **saw blade holder** upwards in the direction of the arrow. Push the **wood saw blade** all the way in, with the teeth facing in front (direction of cut). Make sure the back of the wood saw blade rests firmly in the groove of the **guide roller**.

Fig. 11. A representation of a task step as depicted on the user interface. On the upper screen the user obtains information about the current state of project progress in the form of an overview bar. The user receives a description of the current task in the form of an instruction text in combination with an illustration visualising the instruction. The instruction text contains links (green highlighted words) on specific concepts that should prime the user to ask for further information. For this purpose, a user may either use speech, touch the links or use text input (see text field at the bottom). Furthermore, the user can ask for a video, scheme drawing of the project, or for a more abstract plan by either speech or clicking the appropriate button. In order to navigate through all plan steps the user may use voice commands or click on the respective arrow icons left or right.

For controlling the interaction flow, we implement separate dialogue agents, each responsible for communication with one specific module of ROBERT, i.e. one agent for handling requests related to the planner (e.g. requesting a step-by-step instruction for a specific project), another agent responsible for requests regarding factual knowledge and explanations, and another for handling dialogue-related user input, such as confirmations.

A dialogue agent consists of an individual cloud-based model for natural language understanding (NLU) as well as a dialogue handler, which is able to process module-specific requests. Furthermore, dialogue handlers are also used for keeping track of the dialogue's state as they are aware of each other and obtain knowledge of the interaction's history. The interplay between user interface, dialogue management and the cognitive modules is depicted in Fig. 12.

Dialogue input is forwarded to all dialogue agents by the HTTP API Server in the form of a JSON object containing the converted input text and, if already available, the currently shown task step in the UI. While the currently displayed plan step is used to link input to plan- and dialogue-related context information (e.g. position in the plan, previous user input), the current input text is analysed by the respective NLU module. For NLU we use Microsoft's cloud-based *language understanding intelligent service* (LUIS) [33]. The framework allows creating highly sophisticated and easily extensible NLU models. The statistical-driven approach of LUIS relies on the two trainable concepts *Intent* and *Entity*.

Intent refers to the intention of a user, i.e. the purpose of an utterance, whereas an *Entity* contains meaningful parts of an utterance. For example, when Alice states the following planning request: "I want to build a key rack.", then, the *Intent* would be `startPlanning`, while "key rack" would be recognised as value for the *Entity* named `project`.

In Tab. 2, the developed intents and entities of each model are listed. Intents of each model are designed to be task-independent as they represent genuine elements of ROBERT's dialogue with Alice. The list of all known entities is requested from the ontology manager. This is necessary in order to allow Alice to refer to every concept and individual in the ontology. The dialogue manager itself is developed to be task-independent. After the individual LUIS models have analysed the user input, the response from the model with the highest confidence is selected and transformed into an abstract user act by the respective dialogue agent's handler.

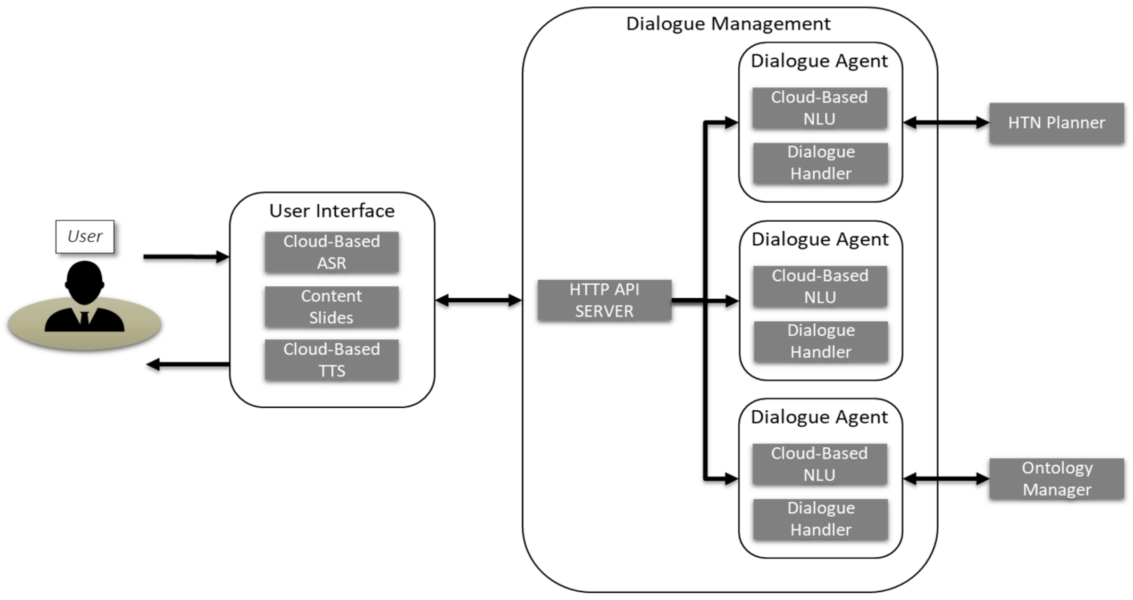


Fig. 12. Modular architecture of ROBERT. The user interacts with the system through a multimodal interface. A dialogue manager consisting of an HTTP API Server and three dialogue agents is used for controlling the interaction flow. While doing so, two agents control the dialogue between user and cognitive modules (HTN Planner and Ontology Manager). Dialogue-related input, e.g. confirmation of information, is processed by an individual agent.

Table 2

The developed intents and entities of each NLU model. *Plan-Related LU* handles planning requests (start planning, abstract or refine the generated plan). In order to request encyclopedic background information (describe) and media content (describeVisual) of a specific ontological concept (saw, battery, etc.) the *Ontology-Related LU* is used. This model is also able to handle availability requests. The *Dialogue-Related LU* handles the affirmation (affirm) or rejection (deny) of previously provided uncertain information.

	Plan-Related LU	***Ontology-Related LU***	***Dialogue-Related LU***
Intents	`startPlanning`	`describe`	`affirm`
	`abstractPlan`	`describeVisual`	`deny`
	`refinePlan`	`doesExist`	
Entities	`project`	`saw`	`saw`
		`battery`	`battery`
		...	...

Dialogue handlers utilise a version of the *Information State* approach [45]. This approach relies on the three concepts *dialogue move* (user input), *information state* (dialogue state) and *dialogue action* (system output).

Depending on the confidence of the agent, a user act is either forwarded to and processed by the appropriate cognitive component (planner, ontology management) or in case of dialogue-related user input, e.g. affirmation of information, directly handled by the intended agent. Affirmation of information is necessary in order to deal with uncertain user input, which may have resulted from poor speech recognition performance. As LUIS classifies intents and entities with a certain probability, there exists the possibility that the classification probability is too low for the system to act in a robust manner.

When the uncertainty about the intent detection is too high, the system runs a confirmation/grounding strategy. In the current implementation we use an explicit confirmation strategy. For example, if ROBERT recognises that Alice wants to have a description of a concept ("What is a metal drill bit?"), but is unsure about the concept (metal drill bit) itself, it would ask the user to confirm the recognized concept ("Do you want to have a description of metal drill bit?"), to which the user can respond with "yes" (affirm) or "no" (deny). Furthermore, the respective agent updates the dialogue state (plan- and dialogue-related context information, e.g. sequence of plan actions, previous user input). The result is then transformed into an abstract system act and returned to the client, where the interface interprets the message and presents the content in visual and/or spoken form.

9. Experimental evaluation

The interaction of users with ROBERT was studied empirically (cf. [46]) with 18 DIY novices (10 females and 8 males, mean age 33.4 years). The task provided to the participants was to construct a key rack from a wooden plank (as shown in Fig. 1) using an electric drill driver and an electric jigsaw while being assisted by ROBERT. To test the potential effect of interactive assistance – as opposed to a more static instruction, such as a pre-fabricated step-by-step guide – the assistant system was provided in two versions:

- Full assistance: the system as described in this paper, with one restriction: verbalisation was only used in the form of automatically generated descriptions (cf. Sec. 6.3), generated explanations (cf. Sec. 6.4) were not offered.
- Baseline: Slides presented instructions at the lowest level of granularity only, using texts (without links for further descriptions, see Sec. 8.2) and images (no videos). Voice commands and question answering were not supported. And no help for tool preparation and operation is provided.

Since our main interest was in studying the interaction with the full assistant, 13 participants were assigned to the full assistance group (six females and seven males) and five to the baseline group (four females and one male).

9.1. Procedure

Demographic data was collected in a pre-test questionnaire together with participants' prior experience with dialogue systems and with DIY. Furthermore, we inquired about participants' help-seeking behaviour in the DIY context and their attitude towards being assisted with DIY projects by an app. Participants in the full assistance condition were shown a short tutorial video introducing how the assistance system can be navigated using speech and touch commands. The DIY task consisted of cutting the wooden plank into two pieces of equal length using the jigsaw, connecting the two parts using screws (and pre-drilling with the electric drill driver), and attaching hangers and hooks. Participants were filmed during the session, and had to fill out a comprehensive post-test questionnaire assessing how they perceived the interaction with the system (as shown in the appendix).

9.2. Results

Pre-test replies indicate that most participants can indeed be considered DIY novices; whereas 15 indicated prior experience with using an electric drill driver, only three had used a jigsaw before, and three had used an electric sander. Two participants indicated no prior experience with any electric DIY tool. The main practical experience reported by the participants was the assembly of pre-fabricated furniture. The main sources of DIY-related help and information were friends/relatives (18), the DIY store (11), youtube (9) and web search (7), but not specialised resources such as literature or DIY forums. Participants indicated that they would welcome being helped by a digital assistance system, as a guide for getting acquainted with a new power tool (average score 4.3 on a five-point Likert scale,[9] 1: disagree – 5: agree) and as a guide to completing a DIY project (average score 4.1). When being asked how difficult the participants judged the task of constructing a key rack, prior to the experiment a mean of 3.1 was obtained (standard deviation 0.85), and after the actual experiment a mean of 3.0 (standard deviation 1.1), indicating a mixed perception of the difficulty of the experimental task.

Within the allocated timeframe of the experiment (105 minutes including introduction, tutorial video in the full assistance condition, and questionnaires), 15 of the 18 participants completed the construction task.

[9]Cf. e.g. Allen and Seaman [47]

Table 3

Mean scores with standard deviations in brackets on a five point Likert scale ranging from 1 (low) to 5 (high).

	Full assistance	Baseline	Total
Navigation & Design	3.75 (0.52)	3.77 (0.93)	3.76 (0.63)
Trustworthiness	4.13 (0.58)	3.80 (0.93)	4.04 (0.68)
Reliability	3.46 (0.78)	3.40 (1.23)	3.44 (0.89)
Predictability & Transparency	3.85 (0.61)	4.25 (1.12)	3.96 (0.77)
Competence	3.74 (0.77)	4.07 (0.68)	3.83 (0.74)
Acceptance	3.69 (0.64)	3.96 (0.74)	3.77 (0.66)
Usefulness & Understandability	3.59 (0.62)	3.47 (1.04)	3.56 (0.73)
Overall evaluation	3.85 (0.69)	4.00 (0.71)	3.89 (0.68)

Three of the 13 participants in the full assistance condition had not finished their work in time and had to be cut short whereas all five participants in the baseline condition completed the task in time. However, this difference between the groups cannot be considered significant. To determine whether this difference is statistically significant (in view of the very small sample size), a nonparametric test such as Barnard's exact test [48] can be used, which is considered more powerful than the popular Fisher's test (cf. [49]). The assumed null hypothesis is that the assignment of the participants to the full assistance vs. baseline condition has no effect on the outcome (i.e. whether participants finish on time). Barnard's exact test yields $p = 0.282$ (two-sided), which means that based on the (admittedly few) data, the null hypothesis should not be rejected (the p-value is larger than any typical significance level to reject the null hypothesis, such as 0.1 or 0.05).

Table 3 shows participants' scores on the dimensions assessed by the post-questionnaire. Since the scores are based on the 5-point Likert scale, five represents the maximal and one the minimal score, with three representing the neutral position. As can be seen, participants mildly leaned towards a positive assessment (on average) when rating the system's navigation and design, its perceived trustworthiness and reliability, its predicatability & transparency, its competence, aspects related to acceptance (cf. appendix), and its perceived usefulness & understandability. When being asked for their overall assessment of ROBERT, the participants' average score fell into the positive sector of the provided scale (3.89). Among the questions in the "Usefulness & Understandability" category, we consider the question whether the participants indicated that they learned something while using ROBERT of particular importance. The participants largely agreed with a mean score of 4.3.

Even though the baseline group is very small, and therefore a comparison of scores warrants caution, it appears that participants in the baseline condition were more favourable of the system's predictability and perceived competence. A possible explanation is offered by the participants' free-text comments on their experience with the system. Five participants in the full assistance condition (out of 13) reported that speech interaction did not work as they expected. Participants were also critical of the assistance offered in the full assistance condition. For instance, some participants indicated that some of the instructions and explanations offered to them appeared too obvious. Some participants were also critical of what they considered technical jargon. Three participants pointed out that they would have expected the shown video clips to be accompanied by an audio commentary (all videos were mute). By contrast, participants in the baseline condition were more critical about the quality of the shown images (not crisp enough) and a lack of detail in the instructions.

Besides the answers provided directly by the participants, we further analysed a more objective performance indicator; the time participants spent to prepare the electric tools for operation. This comprises inserting a battery, an attachment (drill bit, saw blade) and effectuating the settings of the device. Fig. 13 shows the comparison for the two experimental groups (full assistance/baseline). The graph shows a trend of the participants in the full assistance condition of being faster at setting up the drill driver and the electric jigsaw than in the baseline condition. Even though the sample is small, a significant difference in the setup time of the electric jigsaw is observed (a one-way analysis of variance, ANOVA, yields $p = 0.002$). In the case of the drill driver, the difference between full assistance and baseline is less clear. This is not unexpected, however; even novice DIYers can be assumed to be more proficient at using a drill driver (cf. pretest), and are expected to profit less from assistance than in the case of the jigsaw. The overall difference in the time taken for the experiment in the two conditions failed to reach significance. While speed-up itself might not be an important factor for hobbyist DIYers, it can be considered an indicator of learning related to tool handling. Furthermore, in the baseline condition, part of the time spent was observed to result from mishandling of the tools (e.g. inserting the sawblade the wrong way round at first).

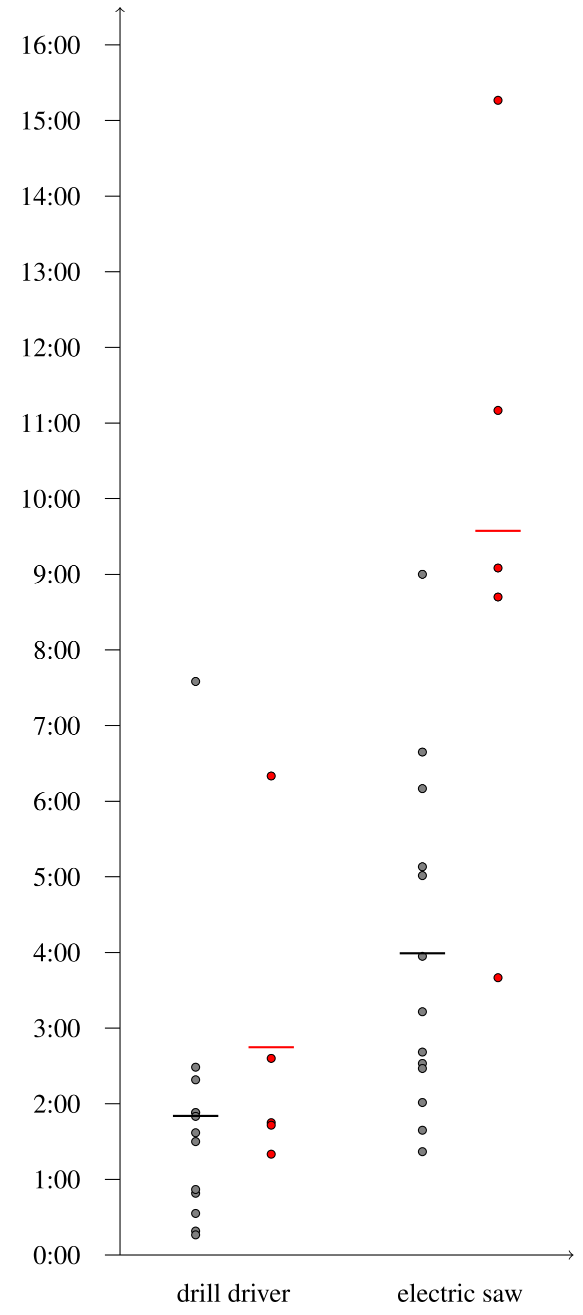


Fig. 13. Scatterplot of time to operation (in minutes) for electric drill driver and jigsaw; left sub-columns: full assistance, right sub–columns: baseline, horizontal bars: mean times.

10. Conclusion and future work

We presented the digital assistant ROBERT, which supports DIY beginners conducting their DIY projects. Assistance is based on the interplay of planning, ontological reasoning, and dialogue management. The DIY projects as well as the involved electric devices are formally represented by means of a planning model and an ontology. Based on these models, a *plan* is developed fully automatically that will, if followed by the user, complete the user's project. This model-based approach allows ROBERT to automatically adapt to previously unknown situations and still provide the best-fitting instructions and explanations in these circumstances. This is especially important in the DIY setting, where the tools and materials available to each individual user differ widely. Furthermore, the model-based approach enables easy maintainability and scalability of ROBERT.

The instructions generated by the planner in the form of a plan are presented to the user in a step-by-step fashion, conveying what to do with the help of images, text, and videos. Suitable media content is retrieved via ontology reasoning, allowing ROBERT to present instructions and media content, even if perfectly suiting material is not available. By exploiting a *hierarchical* planning model, we offer users several levels of abstraction among which they can choose. This way, a user can either be instructed in a very detailed way, showing every single required step, or he or she can be shown abstractions of various steps.

We have conducted a user study to investigate the usefulness of ROBERT. It revealed that the participants were favourable towards the assistance it provides. It also showed that by using ROBERT participants were faster in using electric tools, hinting at an increased proficiency enabled by using ROBERT.

One limitation of the presented user study is that the current incarnation of ROBERT was evaluated as a whole, whereas it would also be informative to test different aspects of the approach separately. On the one hand, the presented evaluation is useful to establish that the integration of the different functionalities serves it purpose. But ideally, separate experiments should be conducted for more closely assessing the impact and/or usefulness of individual functionalities (which, however, generally presuppose a basis functionality on the part of the other components, so that these functionalities can not be considered in isolation). Among the aspects that warrant more focused investigation is that it is not immediately clear to users how they should interact with the system, and to further establish how ROBERT can offer its functionalities in an intuitive way. This also concerns navigation, for instance how to best convey the available hierarchical structure of the generated plans (and corresponding instructions) and how to adjust the presented level accordingly.

Secondly, the presented user study used a small sample of participants (in particular, in the baseline condition). Therefore, the presented empirical results are to be understood in an explorative sense, and the presented statistics should be interpreted only as first evidence.

In spite of the above mentioned limitations, the study provided valuable indications for the future development of ROBERT. There are several lines of future work, most of them are concerned with flexibility of the system and individualisation: At the moment, the system acts in the same way for each user, though there is potential for individualisation. As far as planning is concerned, we will make the level of abstraction shown to the user dependent on the current expertise of the user. That is, a novice user may be instructed on the lowest level of abstraction, whereas a more experienced user may be presented the same plan on a more abstract level. Individualisation can also come into play when choosing the execution order of a plan. A generated plan usually allows for some degree of flexibility when it comes to the order of its actions. For instance, one user could first want to use an electric device with all parts of the project, whereas another might want to use the device only shortly before the respective part of the project is used. So far, we do not take the individual user into account, but plan to do so in the future. We also want to enable users to actively request changes to the plan generated for the project at hand. For instance, in our key rack example, there are many possibilities how to connect the two wooden boards of the key rack. So far, we only offer one, but we could simply add many variants among which the user may choose one (or even combine them). Other preferences could involve the tools, e.g. using screws (and an electric drill) versus nails (and a hammer). All these directions are currently pursued and further empirical evaluations are planned, too.

As far as knowledge modelling is concerned, user modelling can be used to adjust the instructions and explanations for different kinds of users. The need for doing this is also illustrated by the comments of the experiment's participants. Whereas some indicated that they struggled with technical jargon and need more detailed explanations, some considered the information provided by the system as too obvious.

ROBERT currently assumes that whenever an action is performed by the user, it will be completed successfully. This is a suitable assumption for controlled environments, but if ROBERT is used by real-world users, they might fail to perform certain steps. Even simple errors, like inserting a drilling bit wrongly, can have an adverse effect on the success of the overall project. It is also difficult to diagnose the error, as ROBERT has no direct access to the world state after the error has occurred. To be able to assist in these situations we plan to integrate plan repair into ROBERT, which is a capability already included in our previous assistant for setting up home theatres [7]. Using plan repair, ROBERT would then be able to adapt the plan if a failure has occurred so that it still achieves the user's goal.

Acknowledgments

This work is done within the technology transfer project "Do it yourself, but not alone: Companion-Technology for DIY support" 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 thank Juliette Rose and Chunhui Zhu for their assistance with the experimental evaluation.

Appendix. Questionnaires

Navigation and Design

Response format: 5-point Likert-scale (1 = strongly disagree; 5 = strongly agree) except first and second item (1 = very negative; 5 = very good). Cronbach's $\alpha = 0.70$ (in our experimental evaluation)

Table 4
Items of the Navigation and Design Scale (self-developed).

Item	
1	Wie beurteilen Sie die Navigation innerhalb der App auf dem Tabletcomputer (grafische Bedienelemente)?
2	Wie beurteilen Sie die Navigation innerhalb der App auf dem Tabletcomputer (Bedienung per Sprache)?
3	Die Bedienung der App war einfach zu erlernen.
4	Die Gestaltung der App ist ansprechend.
5	Die Gestaltung der App ist Ãijbersichtlich.

Trustworthiness

Translated to German and modified from [50]
Response format: 5-point Likert-scale (1 = strongly disagree; 5 = strongly agree). Cronbach's $\alpha = 0.72$ (in our experimental evaluation)

Table 5
Items of the Trust in Automated Systems Scale [50].

Item	
1	Ich vertraue der App.
2	Die Nutzung der App wird zu gefÃd'hrlichen oder schÃd'dlichen Konsequenzen fÃijhren.
3	Ich misstraue den VorschlÃd'gen der App.
4	Ich bin skeptisch gegenÃijber der App.

Reliability

Translated to German and modified from [51] Response format: 5-point Likert-scale (1 = strongly disagree; 5 = strongly agree). Cronbach's $\alpha = 0.81$ (in our experimental evaluation)

Table 6
Items of the Perceived Reliability Scale [51].

Item	
1	Die App bietet mir immer die Hilfe, die ich benÃűtige.
2	Die App reagiert wie erwartet.
3	Ich kann mich auf eine korrekte Funktion der App verlassen.
4	Die erstellte Anleitung ist stets auf die aktuelle Situation zugeschnitten.

Predictability and Transparency

Translated to German and modified from [51] Response format: 5-point Likert-scale (1 = strongly disagree; 5 = strongly agree). Cronbach's $\alpha = 0.77$ (in our experimental evaluation)

Table 7
Items of the Predictability and Transparency Scale [51].

Item	
1	Ich habe verstanden, wie die App funktioniert.
2	Ich weiÃ§, wie die App auf meine Eingaben reagieren wird.
3	Ich denke, dass mir die App beim Heimwerken behilflich ist.
4	Es ist einfach nachzuvollziehen, was die App tut.

Competence

Translated to German and modified from [52] Response format: 5-point Likert-scale (1 = - -, agree with left statement; 5 = + +, agree with right statement). Cronbach's $\alpha = 0.66$ (in our experimental evaluation)

Table 8
Items of the Competence Scale [52].

Item	
1	Inkompetent - Kompetent
2	Primitiv - Intelligent
3	Laienhaft - Fachkundig

Acceptance

Translated to German and modified from [52] Response format: 5-point Likert-scale (1 = - -, agree with left statement; 5 = + +, agree with right statement). Cronbach's $\alpha = 0.87$ (in our experimental evaluation)

Table 9
Items of the Acceptance Scale [52].

Item	
1	Unangenehm - Angenehm
2	Nutzlos - NÃijtzlich
3	Schlecht - Gut
4	LÃd'stig - Nicht lÃd'stig
5	Unpraktisch - Praktisch
6	ÃĎrgerlich - Erfreulich
7	Starr - Flexibel
8	Nicht individualisierbar - Individualisierbar
9	Passiv - Aktiv

Usefulness and Understandability

Response format: 5-point Likert-scale (1 = strongly disagree; 5 = strongly agree). Cronbach's $\alpha = 0.71$ (in our experimental evaluation)

Table 10
Items of the Usefulness and Understandability Scale (self-developed).

Item	
1	Es war einfach, den Anleitungen zu folgen.
2	Die Handlungsanweisungen waren fÃijr die Durch-fÃijhrung des Heimwerkerprojekts hinreichend.
3	Es konnte stets vermittelt werden, welchem Zweck die einzelnen Handlungsanweisungen dienten.
4	Die App hat sich an meine BedÃijrfnisse und meinen Kenntnisstand angepasst.
5	Die durch die App vermittelten Information zu GerÃd'ten, Arbeitsmitteln und Materialien waren ver-stÃd'ndlich.
6	Durch die Interaktion mit der App habe ich etwas Ãijber das Heimwerken hinzugelernt.

Overall evaluation

Response format: 5-point Likert-scale (1 = strongly disagree; 5 = strongly agree)

Table 11
Items of overall evaluation.

Item	
1	Wie beurteilen Sie die App insgesamt.

References

[1] G. Behnke, M. Schiller, M. Kraus, P. Bercher, M. Schmautz, M. Dorna, W. Minker, B. Glimm and S. Biundo, Instructing Novice Users on How to Use Tools in DIY Projects, in: *Proceedings of the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence (IJCAI-ECAI 2018)*, IJCAI, 2018, pp. 5805–5807.

[2] K. Erol, J. Hendler and D. Nau, Complexity results for HTN planning, *Annals of Mathematics and AI* **18**(1) (1996), 69–93.

[3] M. Ghallab, D.S. Nau and P. Traverso, *Automated Planning and Acting*, Cambridge University Press, 2016.

[4] T. Geier and P. Bercher, On the Decidability of HTN Planning with Task Insertion, in: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, AAAI Press, 2011, pp. 1955–1961.

[5] 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: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, AAAI Press, 2014, pp. 386–394.

[6] F. Honold, P. Bercher, F. Richter, F. Nothdurft, T. Geier, R. Barth, T. Hörnle, F. Schüssel, S. Reuter, M. Rau, G. Bertrand, B. Seegebarth, P. Kurzok, B. Schattenberg, W. Minker, M. Weber and S. Biundo, Companion-Technology: Towards User- and Situation-Adaptive Functionality of Technical Systems, in: *Proceedings of the 10th International Conference on Intelligent Environments (IE 2014)*, IEEE, 2014, pp. 378–381. doi:10.1109/IE.2014.60.

[7] 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: *Proceedings of the 29th AAAI Conference on AI (AAAI 2015)*, AAAI Press, 2015, pp. 4264–4265.

[8] P. Bercher, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nielsen, F. Honold, F. Schüssel, S. Reuter, W. Minker, M. Weber, K. Dietmayer and S. Biundo, Advanced User Assistance for Setting Up a Home Theater, in: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*, S. Biundo and A. Wendemuth, eds, Cognitive Technologies, Springer, 2017, pp. 485–491, Chap. 24.

[9] P. Bercher, F. Richter, F. Honold, F. Nielsen, F. SchÃijssel, T. Geier, T. HÃűrnle, S. Reuter, D. HÃűller, G. Behnke, K. Dietmayer, W. Minker, M. Weber and S. Biundo, A Companion-System Architecture for Realizing Individualized and Situation-Adaptive User Assistance, technical report, Ulm University, 2018. doi:10.18725/OPARU-11023.

[10] M. Schiller, G. Behnke, M. Schmautz, P. Bercher, M. Kraus, M. Dorna, W. Minker, B. Glimm and S. Biundo, A Paradigm for Coupling Procedural and Conceptual Knowledge in Companion Systems, in: *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017)*, IEEE, 2017.

[11] D. McDermott, The 1998 AI Planning Systems Competition, *AI Magazine* **21**(2) (2000), 35–55.

[12] M. Fox and D. Long, PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains, *Journal of Artificial Intelligence Research (JAIR)* **20** (2003), 61–124.

[13] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider and D. Nardi, *The description logic handbook: Theory, implementation and applications*, Cambridge university press, 2003.

[14] K. Myers, P. Jarvis, W. Tyson and M. Wolverton, A Mixed-initiative Framework for Robust Plan Sketching, in: *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, AAAI Press, 2003, pp. 256–265.

[15] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. Chafin, W. Dias and P. Maldague, MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission, *Intelligent Systems, IEEE* **19**(1) (2004), 8–12.

[16] S. Sengupta, T. Chakraborti, S. Sreedharan, S.G. Vadlamudi and S. Kambhampati, RADAR – A Proactive Decision Support System for Human-in-the-Loop Planning, in: *The 2017 AAAI Fall Symposium Series: Technical Reports*, AAAI Press, 2017, pp. 269–276.

[17] J. Fernandez-Olivares, L. Castillo, O. Garcia-Perez and F. Palao, Bringing Users and Planning Technology Together. Experiences in SIADEX, in: *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*, 2006, pp. 11–20.

[18] M.E. Pollack, Planning Technology for Intelligent Cognitive Orthotics, in: *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS)*, AAAI Press, 2002, pp. 322–332.

[19] 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.

[20] C. Steinberger and J. Michael, Towards Cognitive Assisted Living 3.0, in: *Proceedings of the SmarterAAL Workshop colocated with PERCOM*, IEEE, 2018.

[21] R. Stevens, J. Malone, S. Williams, R. Power and A. Third, Automating generation of textual class definitions from OWL to English, *J. Biomedical Semantics* **2**(Suppl. 2) (2011), 5.

[22] S.F. Liang, D. Scott, R. Stevens and A. Rector, OntoVerbal: A generic tool and practical application to SNOMED CT, *International Journal of Advanced Computer Science and Applications (IJACSA)* **4** (2013), 227–239.

[23] I. Androutsopoulos, G. Lampouras and D. Galanis, Generating Natural Language Descriptions from OWL Ontologies: The NaturalOWL System, *Journal of Artificial Intelligence Research* **48** (2013), 671–715.

[24] D.L. McGuinness, Explaining Reasoning in Description Logics, PhD thesis, Rutgers University, 1996.

[25] A. Borgida, E. Franconi and I. Horrocks, Explaining ALC subsumption, in: *Proceedings of the European Conference on Artificial Intelligence (ECAI 2000)*, IOS Press, 2000, pp. 209–213.

[26] T. Nguyen, R. Power, P. Piwek and S. Williams, Measuring the understandability of deduction rules for OWL, in: *First International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM12)*, P. Lambrix, G. Qi and M. Horridge, eds, Linköping University Electronic Press, 2012, pp. 1–12.

[27] M.R. Schiller, F. Schiller and B. Glimm, Testing the Adequacy of Automated Explanations of EL Subsumptions., in: *Proceedings of the 30th International Workshop on Description Logics (DL), CEUR workshop proceedings*, Vol. 1879, 2017.

[28] T.A.T. Nguyen, R. Power, P. Piwek and S. Williams, Predicting the understandability of OWL inferences, in: *The Semantic Web: Semantics and Big Data*, P. Cimiano, O. Corcho, V. Presutti, L. Hollink and S. Rudolph, eds, Springer, 2013, pp. 109–123.

[29] T. Perleth, M. Schiller and B. Glimm, Applying a Model of Text Comprehension to Automated Verbalizations of EL Derivations, in: *Proceedings of the 31st International Workshop on Description Logics*, Vol. 2211, M. Ortiz and T. Schneider, eds, CEUR.

[30] D. Crockford, The application/json media type for javascript object notation (json), Technical Report, 2006.

[31] G. Behnke, D. Höller and S. Biundo, Tracking Branches in Trees – A Propositional Encoding for solving Partially-Ordered HTN Planning Problems, in: *Proceedings of the 10th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*, IEEE Computer Society, 2018, pp. 73–80.

[32] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang, HermiT: An OWL 2 Reasoner, *Journal of Automated Reasoning* **53**(3) (2014), 245–269.

[33] J. Williams, E. Kamal, M. Ashour, H. Amr, J. Miller and G. Zweig, Fast and easy language understanding for dialog systems with Microsoft Language Understanding Intelligent Service (LUIS), in: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL 2015)*, Association for Computational Linguistics, 2015, pp. 159–161.

[34] 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: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, AAAI Press, 2015, pp. 1442–1449.

[35] G. Behnke, P. Bercher, S. Biundo, B. Glimm, D. Ponomaryov and M. Schiller, Integrating Ontologies and Planning for Cognitive Systems, in: *Proceedings of the 28th International Workshop on Description Logics (DL 2015)*, CEUR Workshop Proceedings, 2015, pp. 338–360.

[36] S. Rudolph, M. Krötzsch and P. Hitzler, All Elephants are Bigger than All Mice, in: *Proceedings of the 21st International Workshop on Description Logics (DL), CEUR workshop proceedings*, CEUR Workshop Proceedings, Vol. 353, 2008.

[37] P. Bercher, S. Keen and S. Biundo, Hybrid planning heuristics based on task decomposition graphs, in: *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014)*, AAAI Press, 2014, pp. 35–43.

[38] G. Behnke, D. Höller and S. Biundo, totSAT – Totally-Ordered Hierarchical Planning through SAT, in: *Proceedings of the 32nd AAAI Conference on AI (AAAI 2018)*, AAAI Press, 2018, pp. 6110–6118.

[39] G. Behnke, D. Höller and S. Biundo, Bringing Order to Chaos - A Compact Representation of Partial Order in SAT-based HTN Planning, in: *Proceedings of the 33rd AAAI Conference on AI (AAAI 2019)*, AAAI Press, 2019.

[40] G.A. Miller, The magical number seven, plus or minus two: Some limits on our capacity for processing information., *Psychological review* **63**(2) (1956), 81–97.

[41] R.T. Fielding and R.N. Taylor, Principled design of the modern Web architecture, *ACM Transactions on Internet Technology (TOIT)* **2**(2) (2002), 115–150.

[42] A.S. Rao and M.P. Georgeff, BDI agents: from theory to practice, in: *ICMAS*, Vol. 95, 1995, pp. 312–319.

[43] M.F. McTear, Spoken dialogue technology: enabling the conversational user interface, *ACM Computing Surveys (CSUR)* **34**(1) (2002), 90–169.

[44] M. Kraus, G. Behnke, P. Bercher, M. Schiller, S. Biundo, B. Glimm and W. Minker, A Multimodal Dialogue Framework for Cloud-Based Companion Systems, in: *Proceedings of the 10th International Workshop on Spoken Dialog Systems Technology (IWSDS 2018)*, 2018.

[45] S. Larsson and D.R. Traum, Information state and dialogue management in the TRINDI dialogue move engine toolkit, *Natural language engineering* **6**(3–4) (2000), 323–340.

[46] M. Schiller, G. Behnke, P. Bercher, M. Kraus, M. Dorna, F. Richter, S. Biundo, B. Glimm and W. Minker, Evaluating Knowledge-Based Assistance for DIY, in: *Mensch und Computer 2018 – Workshopband*, Gesellschaft für Informatik eV, 2018.

[47] I.E. Allen and C.A. Seaman, Likert scales and data analyses, *Quality progress* **40**(7) (2007), 64–65.

[48] G. Barnard, Significance tests for 2×2 tables, *Biometrika* **34**(1/2) (1947), 123–138.

[49] C.R. Mehta and P. Senchaudhuri, Conditional versus unconditional exact tests for comparing two binomials, Cytel Software Corporation, 2003, Accessed on January 2nd, 2019 from http://www.nbi.dk/~petersen/Teaching/Stat2009/Barnard_ExactTest_TwoBinomials.pdf.

[50] J.-Y. Jian, A.M. Bisantz and C.G. Drury, Foundations for an empirically determined scale of trust in automated systems, *International Journal of Cognitive Ergonomics* **4**(1) (2000), 53–71.

[51] M. Madsen and S. Gregor, Measuring human-computer trust, in: *11th Australasian conference on information systems (ACIS 2000)*, G. Gable and M. Vitale, eds, Queensland University of Technology, Brisbane, 2000.

[52] J.D. Van Der Laan, A. Heino and D. De Waard, A simple procedure for the assessment of acceptance of advanced transport telematics, *Transportation research. Part C, Emerging technologies* **5**(1) (1997), 1–10.