Artificial Intelligence 247 (2017) 151–169

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Representations for robot knowledge in the KnowRob framework

Moritz Tenorth *, Michael Beetz

Institute for Artificial Intelligence, Universität Bremen, Am Fallturm 1, 28359 Bremen, Germany

ARTICLE INFO

Article history:
Received in revised form 20 May 2015
Accepted 28 May 2015
Available online 3 June 2015

Keywords:
Knowledge representation
Autonomous robots
Knowledge-enabled robotics

ABSTRACT

In order to robustly perform tasks based on abstract instructions, robots need sophisticated knowledge processing methods. These methods have to supply the difference between the (often shallow and symbolic) information in the instructions and the (detailed, grounded and often real-valued) information needed for execution. For filling these information gaps, a robot first has to identify them in the instructions, reason about suitable information sources, and combine pieces of information from different sources and of different structure into a coherent knowledge base. To this end we propose the KnowRob knowledge processing system for robots. In this article, we discuss why the requirements of a robot knowledge processing system differ from what is commonly investigated in AI research, and propose to re-consider a KR system as a semantically annotated view on information and algorithms that are often already available as part of the robot's control system. We then introduce representational structures and a common vocabulary for representing knowledge about robot actions, events, objects, environments, and the robot's hardware as well as inference procedures that operate on this common representation. The KnowRob system has been released as open-source software and is being used on several robots performing complex object manipulation tasks. We evaluate it through prototypical queries that demonstrate the expressive power and its impact on the robot's performance.

1. Introduction

As robotic agents, such as robot factory workers, co-workers, and home assistant robots, scale towards more and more complex tasks in open environments, they will require more flexibility in plan execution. Plans that explicitly spell out every detail of a task only perform well as long as the execution context remains static, but are difficult to adapt in an open, dynamic world. A promising approach for increasing flexibility is to postpone decisions from programming time to execution time by only including the essential aspects of the task in the plan. Humans successfully use this technique when explaining tasks to other humans in terms of vague instructions such as "set the table" or "pour water into the glass". Similarly, instruction sheets that are commonly used in factories and chemical laboratories only briefly summarize the main actions, leaving less important aspects open. The parts that are *not* spelled out are usually those that can be adapted to the situation at hand. While such information gaps can increase overall flexibility, they need to be filled appropriately before the plan can be executed. To this end, a robot needs to supply the "delta" between the information in the (abstract, symbolic and often vague) instructions and the (specific, explicit, and often real-valued) parameters needed by its action components.

* Corresponding author.
E-mail addresses: tenorth@cs.uni-bremen.de (M. Tenorth), beetz@cs.uni-bremen.de (M. Beetz).

http://dx.doi.org/10.1016/j.artint.2015.05.010
0004-3702/

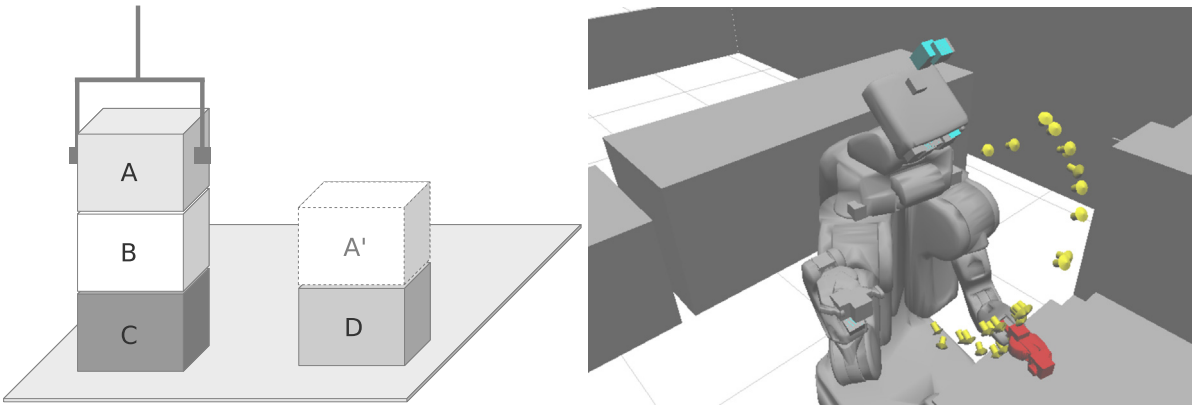

Fig. 1. Left: Example situation in a classical 'blocks world' environment. Using logical inference on the scene representation, the system can determine whether all preconditions are fulfilled for moving block *A* to position *A′*. Right: Visualization of a scene model from the KnowRob robot knowledge base. While the situation is much more complex, efficient algorithms exist for computing e.g. stability or reachability.

This means that it has to interpret the instructions given the current execution context and its background knowledge, and come up with suitable values that can be understood by its motion generation and control components.

This robotics use case requires capabilities of a knowledge representation and reasoning (KR&R) system that differ from what is commonly investigated in AI research. Let us consider the example in the left part of Fig. 1 that shows a typical scene in the blocks world that is still a common scenario in KR research. Researchers in knowledge representation model the scenario as a dynamical system in a knowledge base and approach control, prediction, and analysis tasks by inferring their solutions purely based on this model. The knowledge bases are based on logical [36], STRIPS- or PDDL-based [27,14], probabilistic [18], or other representations. For example, in the logical approach, researchers try to axiomatize the system dynamics, in particular the conditions under which actions can be successfully performed, and the causal laws that relate actions to their effects, such that the solutions of control, prediction, and analysis tasks become the logical consequences of the respective axiomatization.

While these approaches are very elegant as they approach a variety of control problems in a uniform framework, they also typically yield a number of implementation and applicability issues. Since the tasks are solved by inference and search, the computational costs are often high and not predictable. When the physics of the domain become more complicated, the number of axioms grows very fast as we have seen in the axiomatizations of the famous egg-cracking problem [28]. Many approaches thus abstract away from geometric and physical properties that, however, are crucial for robots as they often determine the outcome and feasibility of an action. For the employed coarse-grained discretizations, it is often difficult to find the appropriate level of abstraction for continuous quantities which are pervasive in robotics applications. Once abstracted, information cannot be accessed on a finer level of detail. Examples of inferences that depend upon such continuous quantities are the reachability of objects and the stability of object placements [7].

The right part of Fig. 1 visualizes how the scene of the block-stacking task may more realistically look like for the robot. It can easily be seen that it is significantly more complex than the blocks world model, making the application of many AI reasoning methods difficult. However, all of this information is already present in the robot's internal data structures: the robot's pose and movements, recognized objects, environment structures, etc. And for many inference tasks, such as the visibility of objects or the reachability of locations, there are efficient and well-understood algorithms in robotics that operate on these data structures. Reachability can be computed by inverse kinematics calculations [38]. Visibility can be computed by rendering the scene from a given viewpoint [29]. These algorithms do not require abstraction into a symbolic knowledge base, and by operating on the original data, can often compute much more detailed results than logical inference.

In this paper, we argue that instead of abstracting the data into a purely symbolic knowledge base, robots shall rather keep the original, high-resolution, continuous data and only compute a "symbolic view" on it as needed. This on-demand abstraction can always provide the appropriate level of detail: Robotics algorithms can operate on the original data, logical inferences can be computed on a more abstract symbolic level. Queries can combine inferences made at all levels of abstraction.

This is the approach taken by the KnowRob knowledge processing system that we present in this article. It consists of a (rather shallow) symbolic knowledge base that provides semantic representations of the robot's data structures that primarily serves as integration layer for various inference algorithms. The representation can be shallow compared to expressive AI reasoning methods since few inferences are done at this level. For example, it does not have to model the appearance of objects if the visibility reasoning algorithm can read it from a linked 3D model. And it does not have to model physical laws if an algorithm for computing object stability can read its physical properties from a CAD model and parameterize a simulator with them. The role of a KR system therefore shifts from a world model that has to be consistent and complete towards a semantic integration layer. The correctness of inferences is then determined by the correctness of the algorithms

rather than a complete and correct axiomatization that would anyway be difficult to guarantee if information is read from noisy sensors.

The remainder of this paper is organized as follows. We start with an overview of lessons learned while investigating knowledge-enabled robot control and explain how the findings have been implemented in the KnowRob system. The following section explains the representations of objects, their parts, properties, and locations in the environment as well as the robot's self-model. It is followed by the representations of events and actions at the instance and class level and methods for projecting the effects of actions on objects. We evaluate our approach by the knowledge content, scalability, example queries showing inferences it enables, examples of robot experiments and its adoption by the open-source research community. The paper finishes with a comparison with related approaches and our conclusions.

While parts of the KnowRob system have been described in several earlier (conference) papers, this article tries to set these individual aspects in relation and to explain how they contribute to the overall representation and reasoning capabilities. It is accompanied by open-source software, ontologies and knowledge bases.[1] Several online tutorials on simulated and real robot data explain how the system works and what it can be used for. In addition, we have created a cloud-based version of the KnowRob knowledge processing system called OPEN-EASE.[2] Users can create a free account and try the system without installation by sending queries via a web browser. By providing the software and representations, we would like to give the reader the opportunity to try the system and assess its performance for different applications.

2. Lessons learned about robot knowledge processing

In the introduction, we have argued that there are important differences between knowledge processing methods for robots and the topics commonly investigated in KR&R research. In this section, we would like to summarize our insights and lessons learned while implementing the different versions of the KnowRob system, and motivate the design decisions that have been influenced by them.

2.1. No fixed levels of abstraction, no layers, no "black boxes"

A core observation is that a KR&R system for robots has to consider both continuous, subsymbolic data and symbolic knowledge, and support both abstraction and concretization. However, we found that any choice of one or more fixed levels of abstraction turned out to be inappropriate for use cases that either needed coarser or more detailed models. Especially the concretization of abstract information is not well supported in most approaches that treat the underlying layers as "black boxes". We therefore chose to store information in the most detailed format available, and abstract to the appropriate level when this information is needed for inferences. Besides providing information at the right level of abstraction, this approach also alleviates the problem that the symbolic knowledge deviates from the continuous-level data since the former is computed from the latter at query time.

2.2. A knowledge base should reuse data structures of the robot's control program

The content of a robot KR system has to be continuously updated with new perceptual or actuation data. As this information is already present in the robot's control system, we consider the KR system as a kind of 'parasite' on top of the existing data structures whose task is to make sense from them and to lift them to a conceptual level. As a consequence, the content of the knowledge base is grounded in these data structures that have carefully been developed by a robot programmer in such a way that they are grounded in the outer world, for instance using specialized perception routines. If they were not grounded, the robot would not be able to perform its tasks. This approach differs from other AI knowledge bases that directly refer to entities in the outer world, making the grounding problem more difficult.

2.3. Symbolic knowledge bases are useful, but not sufficient

Symbolic representations are undeniably very useful for structuring the knowledge base by providing a "type system" and for performing logical and qualitative inference. However, the level of detail and the many types of information required for successfully accomplishing robot tasks, such as exact times, 3D geometric information, kinematic structures and appearance models, are difficult to encode in purely symbolic form. A proper representation of the robot's complex dynamic surroundings would require very expressive formalisms, while the fact that much information is obtained from partial observations with noisy sensors will inevitably lead to wrong and inconsistent information that is difficult to resolve in a purely logical knowledge base.

Our knowledge base only has a rather shallow symbolic representation of general concepts, while most information about the robot, its environment and perceptions is merely a "virtual knowledge base" computed at runtime from the data structures of the robot's control program. In a way, this approach is in line with Brooks' observation that "the world is its

[1] Available for download at http://www.knowrob.org.
[2] http://www.open-ease.org.

own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough" [6]. As a consequence, much less reasoning is done at the symbolic level, which relaxes the requirements on the correctness, completeness and consistency of the knowledge base. We thus cannot guarantee either of them, but due to the structure of the knowledge base, this is much less of an issue than in other systems.

2.4. Robots need multiple inference methods

One query to the knowledge base may require the use of different inference methods and different information sources. For example, a robot may ask if its gripper has been inside a container during a pick-up task in order to verify that a pick-up action has been attempted. Such a query requires temporal reasoning about events during the pick-up action, ontological reasoning to determine which objects are containers and three-dimensional spatial reasoning to determine the "inside" relation. These reasoners need access to the robot's task logs, to proprioceptive information about the gripper movements, to the environment model and to general knowledge about types of objects. We therefore decided to use a common underlying representation for all these kinds of information and to have an ensemble of expert reasoners operate on this shared knowledge. This approach is similar to other ensemble-of-experts architectures such as the IBM Watson system [11] that has demonstrated impressive performance on real-world question answering not by using a single reasoning method, but a collection of several specialized ones. In contrast, many classical AI reasoning methods focus on either temporal or qualitative spatial or ontological reasoning, while not supporting the other ones.

2.5. Evaluating a robot knowledge base is difficult

There is a wide range of aspects that contribute to the performance of a robot knowledge processing system, including the number of facts it contains, the coverage of relevant knowledge areas, the range of inference methods that are supported, the run time for common queries, the scalability regarding the amount of information stored, the usefulness of the computed results for robot tasks, etc. However, few of these aspects can easily be quantified, and some such as the runtime for answering queries depend on factors such as the size of the knowledge base or even the way how the query is formulated. While we try to evaluate as many of these aspects as possible, exhaustively covering all of them is usually beyond the scope of a paper. However, since the KnowRob system has been available as open-source software for a while, we consider the adoption by the community and the novel application areas it has been used in by external developers another strong indicator of its quality. In addition, we have developed a web-based version of the knowledge base that users can try without requiring any installation to evaluate its usefulness for their particular use case.

3. The KnowRob robot knowledge processing system

The concepts explained in the previous section have been implemented in the current version of the KnowRob robot knowledge processing system. In an earlier paper [43], we have introduced KnowRob with an emphasis on *systems aspects* such as the knowledge storage and techniques for integrating the knowledge base with the robot's data structures, perception and control system. In this article, we focus on its *representational structures* and how they contribute to its reasoning capabilities.

Since KnowRob combines information of many types from different sources and several inference methods, their integration becomes an important issue. This includes both the integration at the representational level, to ensure that all modules share a common language, and the integration at the programmatic level, to ensure that the right inference methods are called when processing a query.

We approach the former by representing all knowledge in the system with respect to a common ontology. The structures presented in the following sections have been implemented in the Web Ontology Language OWL [50], a standardized language based on Description Logics. The core of the knowledge base is formed by a large ontology that conceptualizes the robotics domain. Its upper levels have been derived from OpenCyc [23], a widely used and comprehensive upper ontology. Staying compatible with OpenCyc allows us to incorporate extensions made by other researchers, for example mappings to other ontologies like WordNet [10]. The general-purpose upper ontology can be extended with *micro-theories* that add domain-specific knowledge or special-purpose inference methods. Compared to other knowledge representation formalisms, OWL has rather shallow semantics and only allows few kinds of inferences to be made on the relations between classes and individuals. As OWL inference would not be sufficient as the only inference method, KnowRob allows to complement it with other reasoning algorithms. They do not have to be of logical nature, but can employ any kind of computation as long as they read their input data from the knowledge base and produce facts formulated in the common representation language. OWL is thus primarily used as "glue" at the representational level, as a common and structured language for describing the world.

For integrating the different modules at the programmatic level, we use the logical programming language SWI Prolog [53]. The OWL ontologies are loaded into the system using SWI Prolog's Semantic Web library [52], representing their triple structure as Prolog predicates. Since Prolog has both declarative and procedural semantics, it allows to combine this logical knowledge base with programming: Using Prolog's Foreign Language Interface, we can integrate external reasoning tools and also expose them as Prolog predicates. For example, we could create a predicate *reachable(O, L, R)* which describes

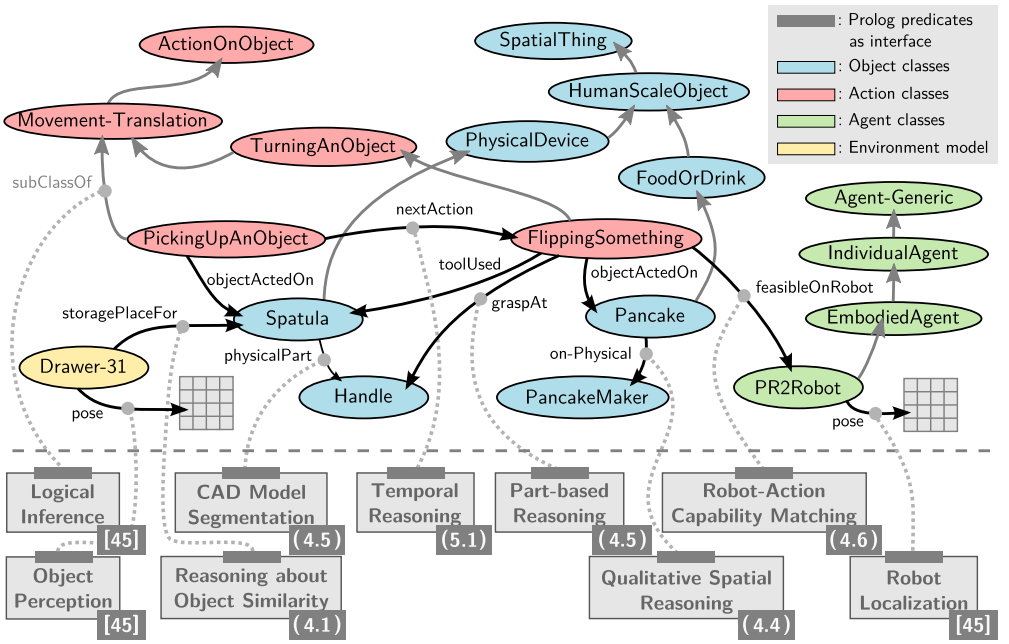

Fig. 2. Excerpt from the knowledge base, describing information around two actions for picking up a spatula and flipping a pancake. The upper part describes the OWL model, with the different interacting knowledge domains highlighted in different colors. The lower part lists some of the reasoning methods that are hooked to the classes and relations in the OWL model they compute. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

that an object *O* can be reached from a location *L* by the robot *R* that is, behind the scenes, evaluated by computing a solution to the underlying inverse kinematics problem. This common interface allows to combine inferences in a single Prolog query, for example to ask for all objects of type *Cup* that are reachable by the robot *PR2* from its current location *L*:

```
?- owl_individual_of(Obj, kr:'Cup'),    % computed by OWL inference
   robot_loc('PR2', L),                 % computed by querying the robot's localization system
   reachable(Obj, L, 'PR2').            % computed by inverse kinematics calculations
```

Efficiency is of key importance for robots since results have to be available fast enough to still be relevant for the current actions and to not slow down the task execution. To achieve efficient and scalable inference, we can employ specialized inference techniques (which exploit the structure of a problem to become efficient) or simply implement inferences procedurally that would be very difficult or even intractable using pure logical reasoning. In many cases, we have found that the search trees for our queries become rather flat, which lets them resemble Datalog programs that do not require a deep search for solutions and can therefore be answered very efficiently.

3.1. Example: integration of inference methods with the OWL knowledge base

Fig. 2 illustrates how external reasoners are integrated into the OWL model that serves as conceptual framework for storing the information they produce and consume. It visualizes a small excerpt of the knowledge base, representing a sequence of two actions in a plan for making pancakes. The upper part of the figure describes the conceptual model of this problem as an OWL model. The different colors indicate that already this small example requires the combination of knowledge from different areas. The gray arrows in the figure visualize the *subClassOf* relations, black arrows denote other relations like the *objectActedOn* of an action. The first action, *PickingUpAnObject*, is performed on an object of type *Spatula* which is stored in *Drawer-31*, an instance of a drawer at a specified pose (i.e. position and orientation). *PickingUpAnObject* is a subclass of the class *Movement-Translation*, which by itself is a specialization of *ActionOnObject*. OWL classes inherit properties from their parents, which is a powerful tool for representing knowledge in a generic way. The second action, *FlippingSomething*, is linked to the first one using the *nextAction* relation, allowing inference about the sequence and temporal aspects. The *Spatula* serves as *toolUsed* in this action and is *graspedAt* the *Handle*, which is one of its *physicalParts*. The flipping action is performed on a *Pancake* that is located on top of a *PancakeMaker*. Before execution, the system has checked whether the actions are *feasibleOnRobot* on the *PR2Robot* to make sure that all required capabilities are available.

While these structures could be defined manually in OWL, they will usually be populated by a variety of automated reasoners that are 'hooked' into the OWL representation to compute relations or fill the conceptual view with content. In the example, the relations with a gray circle are not asserted manually, but computed at query time by the inference tools

in the lower part of the figure. Some reasoners can not only compute a relation between existing individuals, but also create new individuals, for example based on perception results, and integrate them into the knowledge base.

4. Representation of objects and spatial information

This section introduces the representations and associated inference methods for objects and other spatial information. We start with class-level knowledge about objects types, continue with object instances, the representation of positions and orientations, the computation of qualitative spatial relations, and the representation of the object geometry and their composition from parts. The last subsection deals with special kinds of objects, namely parts of the robot and their properties.

4.1. Class-level object knowledge

Object type information is organized in a large ontology that ranges from very generic classes such as *SpatialThing* to specific ones such as *Refrigerator–Freezer*. In total the KnowRob ontology contains about 7000 object classes. Whereas the upper levels have been manually imported from OpenCyc to obtain a sound basis for the ontology, many of the more specialized classes have been generated automatically from existing information. For instance, we have explored how information from online shops, in particular the category structure, can be translated into a class taxonomy [44] to facilitate the creation of large-scale ontologies.

The classes are annotated with properties that describe for instance that an *Oven* has *HeatingFood* as its primary function and a *Handle* as *properPhysicalPart*. Properties are inherited by sub-classes, and by inheriting from multiple super-classes, different facets of an object can be described: A *Refrigerator*, for example, is a subclass of both *Box-Container*, *Electrical-HouseholdAppliance* and *RefrigeratedStorageDevice*, and inherits the properties of all of these classes. This way, objects can be classified along different dimensions (e.g. shape, need for electricity, temperature), and rules can be formulated in a generic fashion by selecting the appropriate superclass.

Compared to other approaches, for example [15], the KnowRob ontology is rather weakly axiomatized and contains fewer class restrictions that define concepts. For robots operating in an open world, definitions such as “A living room is a room with exactly one sofa and at least one TV” often turn out to be too brittle since a living room without a TV or with two sofas would cause a logical inconsistency. Inference about such fuzzy or uncertain relations is therefore performed by integrated probabilistic reasoners [32] instead of description logics.

The two most common reasoning tasks at the object class level are the selection of all subclasses of a generic concept (e.g. all *Container* objects) using common OWL inference, and the computation of a semantic similarity between classes. In [37], we have shown that a similarity computed as the weighted distance of two concepts in the ontology can for example be used to reason about object locations: Objects are often stored at places where semantically similar objects are located. The following equation computes the WUP similarity [56] from the depths of two concepts C_1 and C_2 and their least common superconcept in the ontology:

$$wupSim(C_1, C_2) = \frac{depth(LCS(C_1, C_2))}{\frac{1}{2}\,(depth(C_1) + depth(C_2))} \tag{1}$$

4.2. Object instances and environment maps

The robot’s environment is represented using instances of these object classes. Instead of directly representing the objects in the outer world, the OWL individuals are merely considered as a description of them. Objects are thus referenced by their properties, not their identity, and multiple (possibly partial) descriptions of an object can exist. The descriptions, called “designators” [26], can be redundant (e.g. if an object has been detected by different methods), incomplete (e.g. if something has been detected, but its type is not yet known) or inconsistent (e.g. in case of perception failures). This is very useful to distinguish the time when an object has been created and when it was first detected, to maintain a belief about objects that do not exist any more, and to unify multiple instances once the robot finds out that they refer to the same object in the real world. This is supported by the OWL language that does not make the Unique Name Assumption, but rather provides constructs (*owl:sameAs*, *owl:differentFrom*) for describing whether or not two identifiers refer to the same object.

Any set of object instances can be associated with an instance of a *SemanticEnvironmentMap* (Fig. 3 left) that may additionally store map-level information such as the address of the space described by the map [33,41]. Object instances can be composed in a part-of hierarchy to describe the composition of complex objects or kinematic structures such as robot parts or kitchen cabinets (Fig. 3 right). Appearance models or 3D surface mesh models can be associated with each subcomponent (see also Section 4.5). Since there is no structural difference between static objects in an environment map, movable objects detected by the robot’s perception system, and parts of the robot, any kind of object information in the system can easily be compared.

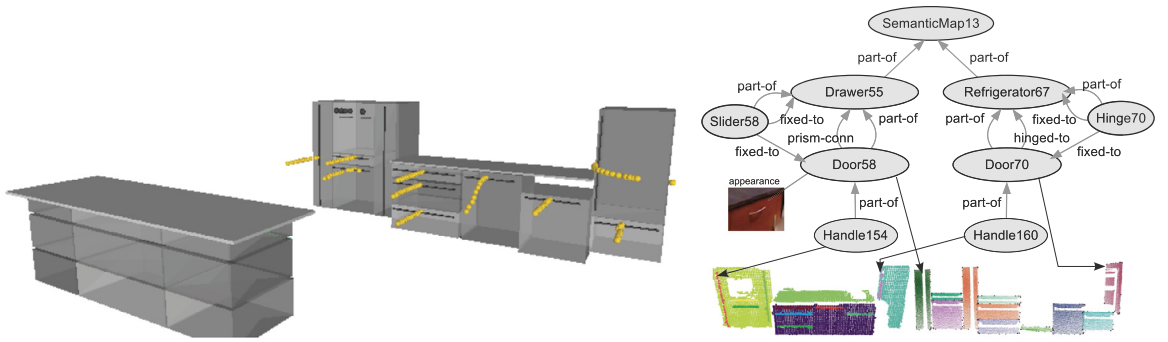

Fig. 3. Examples of semantic environment maps. Left: Map of a kitchen including trajectories for opening the different cupboards. Right: Representation of composed objects and kinematic structures with prismatic and rotational joints.

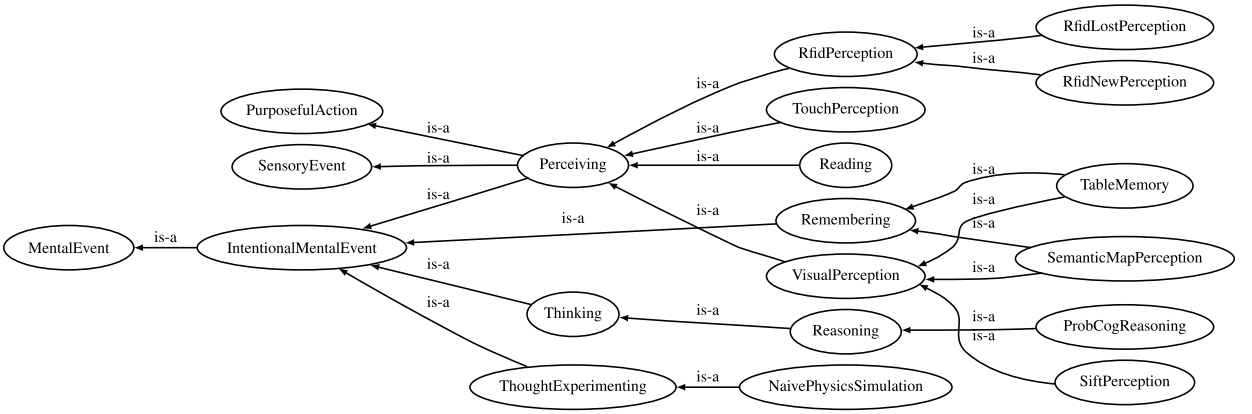

Fig. 4. Ontology of mental events. Each of these events can result in changes in the robot's belief state about object poses.

4.3. Representation of positions and orientations

Among the most important information for a robot are object poses (i.e. their positions and orientations) and their changes over time. In KnowRob, pose values are stored as a 4×4 matrix that encodes both translations and rotations with respect to a given coordinate system (with the coordinates of the environment map as a default). While this is fairly common, representing how the values change over time is a bit more difficult. On the logical level, it requires the ability to qualify a relation (in this case the *atLocation* property that links an object instance to a pose) with the time when it was valid, turning it from a binary relation between the object and the pose to a ternary one that also includes the time. While ternary relations are not directly supported by OWL, they can be converted into a set of binary relations by introducing a first-class object with binary links to all entities.

On a semantic level, we would like to represent the type of belief about an object's position, e.g. if an object has been perceived or if the robot just expects to find it there. We therefore use the (typed) event that has lead to the belief about an object pose as the reified relation object. For example, if an object has been perceived using the robot's camera, we create an instance of the class *VisualPerception* that is linked to the object, its pose and the time point. This structure is visualized in the left and right parts of Fig. 5. These events are subclasses of the class *MentalEvent* as visualized in Fig. 4. Multiple events can be assigned to one object, either for storing several detections over time, or (possibly differing) poses determined using different methods. This representation allows to reason about changes over time (previous world states can be reconstructed by only considering events before a given time) and about discrepancies between e.g. the desired and the perceived poses.

The representation is similar to the fluent calculus [47] in which fluents are data structures that represent the change of values over time. In our case, however, the reified objects are more than just a changing value because they provide a memory of past states, they describe the source of this relation by their type, and allow to reason about multiple "possible worlds", for example the perceived, desired and simulated world.

4.4. Reasoning about object poses: computation of qualitative spatial relations

Humans usually describe locations in qualitative terms (e.g. "on the table") instead of exact coordinates, which can also be useful for robots, for example to formulate more generic rules (e.g. "all objects on the table shall be put into the

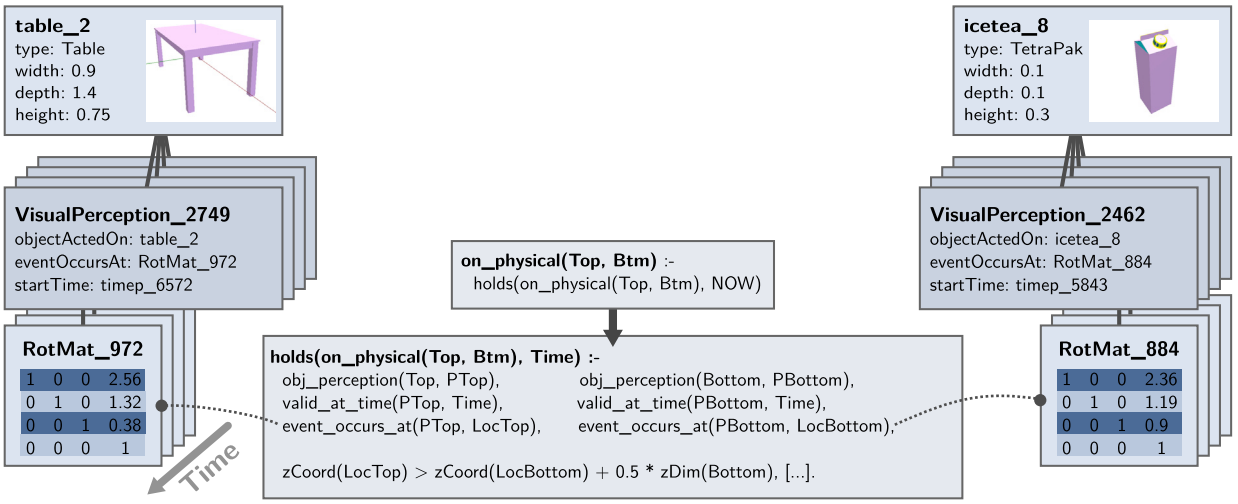

Fig. 5. Reasoning about the KnowRob object representation for computing qualitative spatial relations. The *holds* predicate computes the *on_physical* relation by retrieving the latest detections of the two objects in the relation before a given point in time.

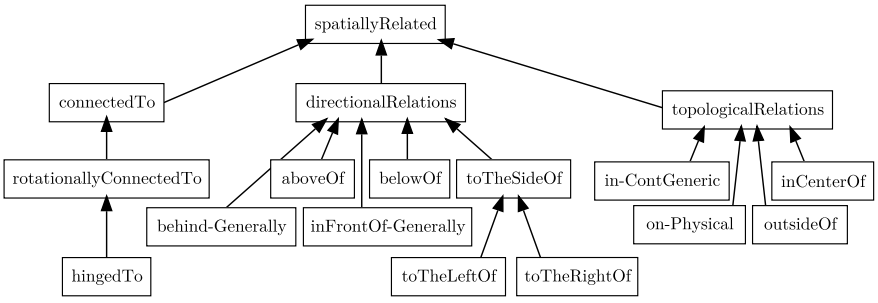

Fig. 6. Taxonomy of qualitative spatial relations including directional and topological relations.

refrigerator"). In this section, we show how such qualitative relations can be computed from the spatio-temporal object pose representation introduced in the previous section. The computation is done on demand by inference procedures that are attached to the OWL properties they compute, which avoids the storage of an exploding number of pair-wise relations as well as the frame problem of determining which relations have to be updated when an object has been moved.

The representation of object poses using *MentalEvents* provides the information for evaluating qualitative spatial relations between objects over time. We assume that the last perceived pose of an object is still valid at the time the relation is evaluated, which is the case if no other agents or self-moving objects exist. In general, a relation *rel* between *A* and *B* that is true at time *T* is expressed in KnowRob by the *holds(rel(A, B), Time)* predicate. Fig. 5 illustrates its implementation in the box in the lower center that accesses the detections of the two objects *table_2* and *icetea_8* shown at the left and right edges. The first step is to search for perceptions of the top and bottom objects that were valid at *Time*. If no explicit end time is set for a perception, it remains valid until the current time or until another perception of the same object has been made. The procedure searches for perceptions with a validity interval that contains *Time* and reads the poses where objects have been perceived in these events using the *event_occurs_at* predicate. By comparing the object poses and (optionally) their dimensions, all relations in Fig. 6 can be computed. For the "on top of" property, one has to check whether the *z* coordinate of the top object is larger than that of the bottom object and compare the object dimensions (omitted in this example). Since the majority of queries are concerned with the current world state, KnowRob offers a simplified query mechanism that uses the current time as default (upper block in Fig. 5).

4.5. Object geometry and functional parts

In particular fine manipulation or tool usage require detailed geometric information about the objects involved that is difficult to represent in a symbolic knowledge base. We therefore extend the object classes with links to detailed three-dimensional CAD models, which provide a very detailed geometric description and are often available from free databases on the Web. While such monolithic CAD models are already useful, e.g. for visualization or grasp planning, robots often need to interact with specific object parts that have functional meaning for the action at hand: For picking up items, they

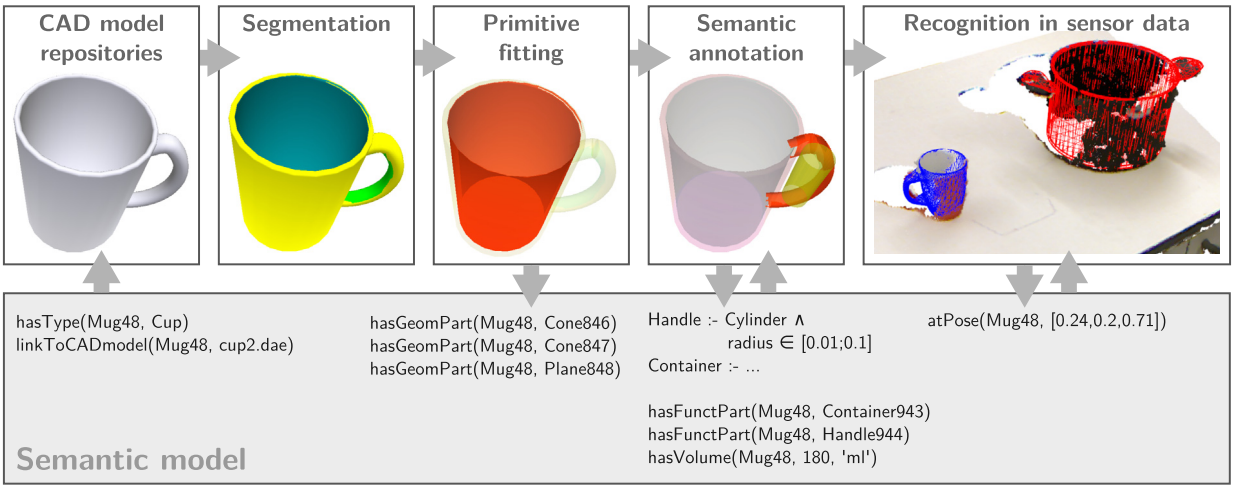

Fig. 7. Proposed method for generating the part-based object representation. A three-dimensional CAD model of an object is segmented and its parts are classified into functional categories based on semantic rules in the knowledge base.

should use the handle; for pouring something from or into a container, they need information about its opening direction and volume; for pouring something onto a surface, they have to select a suitable horizontal surface.

These inferences require a tight integration of geometric and semantic information, which we again implement as a "virtual knowledge base", i.e. a conceptual model with attached procedures for computing the relations, on top of the 3D models. The virtual symbolic "view" on the model can be generated at runtime and allows to answer logical queries about the geometric model and its functional components. The structure of these part-based model is exactly the same as for other composed objects such as cupboards (as shown in Fig. 3 right), but each instance of a sub-part such as a *Handle* or a *Container* is grounded in a (sub-)mesh of the CAD model. Once the objects have been recognized in a scene, the object-relative poses of the identified parts can be translated to scene-global positions.

In [46], we have described how such a model can be created automatically from a CAD model by linking geometric analysis methods with knowledge-based definitions of these components. Fig. 7 outlines the main processing steps of the proposed approach: The model is segmented based on the surface curvature, and geometric primitives such as cones, spheres and planar surfaces are fitted to the segments. After these bottom-up processing steps, the system can apply knowledge about functional parts in a top-down fashion. It uses logical rules that define the components in terms of geometric primitives in order to identify these functional object parts. For example, a handle can be defined in the knowledge base as a cylinder of certain dimensions, or a container can be formed by a concave cylinder that is closed by a planar surface at one end. The rules are composable, so definitions of other kinds of handles can easily be included by adding appropriate rules. The resulting models allow the evaluation of logical queries on geometric object models and to reason about functional parts at both a geometric and semantic level.

4.6. Robot components and capabilities

Besides models of the outer world, robots also need representations of their own structure and capabilities: They allow a robot to reason about which components it consists of, which properties they have, how they are (kinematically) connected, and which capabilities they enable. This can for instance be useful to determine if the robot is lacking components that are required for a given action. While commonly used robot description formats such as URDF[3] or COLLADA [2] provide information about the kinematics, dynamics and 3D surface of a robot, they lack semantic information about its parts. For example, they do not represent which of the components are sensors, or which group of components forms a gripper. In KnowRob, robots are described using the Semantic Robot Description Language (SRDL) originally proposed in [21] that extends these low-level robot models with semantic representations. At the lowest level, the robot's hardware is (like in URDF) described in terms of links and joints (Fig. 8 left). This representation can automatically be generated from a URDF robot model and is again very similar to the models of composite objects in general (Fig. 3). At a higher level, the links and joints can be aggregated to semantically meaningful components such as arms and hands (Fig. 8 right). Also, links that correspond to sensors or actuators can be annotated with properties, for instance the field of view of a camera or the resolution of a laser scanner. On top of the component model, SRDL describes abstract capabilities such as "navigation" or "object recognition" and their dependencies on components. At the highest level, actions can define dependencies on components (e.g. a camera with certain properties) and capabilities (e.g. recognition of textured objects). These dependencies

[3] http://www.ros.org/wiki/urdf.

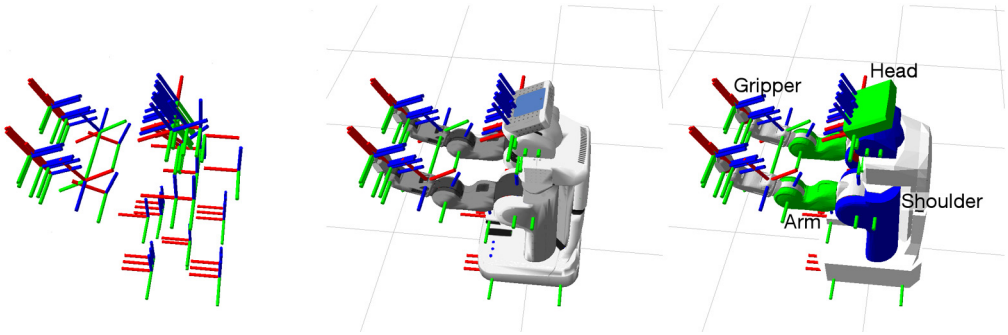

Fig. 8. Typical robot descriptions cover the kinematic and dynamic aspects (left) as well as surface models of a robot (center). A semantic robot description further adds information about the meaning of the different robot parts (right).

can automatically be checked against a robot model to infer about whether all components required for the action are available. In addition, the component model can also be used for other kinds of reasoning, for example to compute which camera can see an object based on its pose and field of view [54].

5. Representation of events and actions

In addition to models of their spatial context, robots need representations of temporal events and, importantly, of the actions they perform. In KnowRob, instances of observed events and performed actions form an episodic memory of the robot's experiences. We will introduce their representation in Sections 5.1 and 5.2. Action models at the class level describe abstract plan schemata that can for example be generated from instructions given by humans. Gaps in these descriptions can be filled using planning techniques. In Sections 5.4, we explain how action effects are represented and reasoned about in KnowRob. In the end of this section, we discuss how action models can be extended with models of processes that are triggered as side-effects of actions and may influence their results.

5.1. Events and temporal information

Temporal information such as the start time of an event, the duration of an action, or the contemporaneity of two events is highly important for robots that act in dynamic environments. Similar to OpenCyc, we consider an event as *"a dynamic situation in which the state of the world changes"*.[4] Events can be instantaneous (such as the moment when a perception is made) or temporally extended, such as the execution of a trajectory for reaching towards an object. Each event has a *startTime* and, if its duration is finite, also an *endTime*. Both relations link an event to a *TimePoint*, a special case of a *TimeInterval* with zero duration. This event representation, visualized in Fig. 9, serves as basis for temporal reasoning using time interval logics such as Allen's interval algebra [1]. Similar to the methods for reasoning about qualitative spatial relations introduced in Section 4.4, the relations between time intervals are implemented as Prolog predicates that read information from the OWL-based model. Actions are considered as a special kind of event, so the same representations and inference methods can be used for both actions and general events.

5.2. Action instances

Action instances represent actions that have been performed in some way – either by the robot itself in the real world [54], as part of an envisioning procedure [42], or by a human demonstrating them to the robot [4]. While some of these models will contain more information than others (for example, the goal of an action can be logged by the robot, but cannot be observed for human actions), all share the same basic structure visualized in Fig. 9. This common language makes it easy to combine or compare action information from these different sources. Each action has a *startTime* and *endTime*, in addition to further information on its parameters such as manipulated or perceived objects. Hierarchical task structures can be built up by the *subAction* relation. Besides the actions that are part of the task tree, asynchronous events such as sensor readings can be stored using the same mechanisms and be related to simultaneous actions using the temporal reasoning methods introduced in the previous section.

Again, this abstract symbolic model is complemented with continuous-level information, for example about the trajectories of robot movements as visualized in Fig. 1 (right). During task execution, detailed log files of the movements performed by the robot or human are recorded in a high-volume database [54]. Based on the start and end times, query predicates can later read poses of body parts at semantically described time points, e.g. the end of a reaching motion, or time intervals [5].

[4] Part of the definition of an event in OpenCyc http://sw.opencyc.org/concept/Mx4rvViADZwpEbGdrcN5Y29ycA.

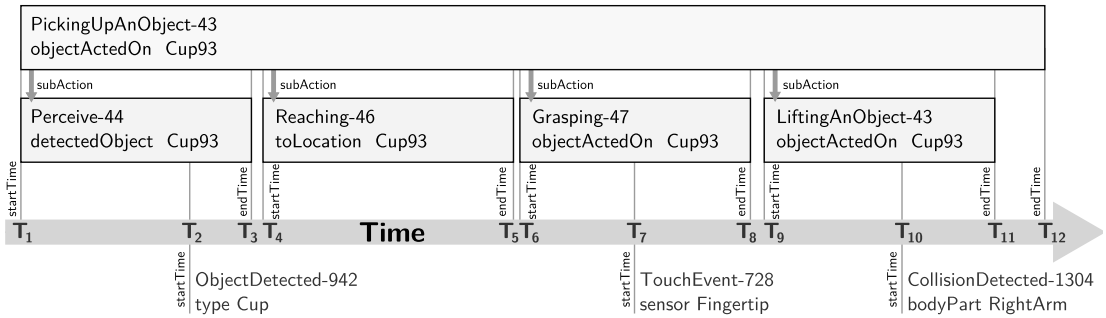

Fig. 9. Example time line of events, consisting of some action instances with extended duration and some instantaneous events. Various temporal relations can be computed based on the start- and end times.

5.3. Action classes

Similar to the ontology of object types, the action ontology, which contains about 130 action classes, mostly serves for providing the vocabulary for representing tasks. In comparison, there are more class restrictions that describe the actions' preconditions and effects as explained in Section 5.4.1. We make use of the taxonomic structure for generalization by inheriting restrictions from their superclasses.

Robot plans are modeled as "action recipes" that abstractly describe the composition of tasks from sub-actions, the types of these actions, their relative ordering, as well as action parameters e.g. which object is to be manipulated or at which locations the action is to be performed. These action models have originally been developed as "action recipes" as part of the RoboEarth language [45] for the exchange of task descriptions between robots and has been extended to other use cases since. The description of a concrete task usually derives specific sub-classes from the general action classes in the KnowRob ontology and annotates them with task-specific information about objects, tools, locations or timing. These derived action classes are then arranged in a (partially) sequential order. The example code below is an excerpt of a plan for setting a table.

```
Class: SetATable
    Annotations: label "set a table"                # Natural-language label, often the command
  SubClassOf: Action                                # given by a human
  EquivalentTo:
      subAction some PutPlaceMatInFrontOfChair      # References to task-specific action classes
      subAction some PutPlateInCenterOfPlaceMat
      subAction some PutKnifeRightOfPlate
      subAction some ...
      orderingConstraints value ...                 # Partial ordering constraints among the subactions


Class: PutPlaceMatInFrontOfChair                    # Extension of the generic class
  EquivalentTo:                                     # PuttingSomethingSomewhere with task-specific
      PuttingSomethingSomewhere                     # properties (objectActedOn, toLocation)
      objectActedOn value PlaceMat1
      toLocation some Place1

Class: Place1                                       # Action arguments can be nested class restrictions,
  EquivalentTo:                                     # e.g. for describing places by their properties
      inFrontOf-Generally some Chair-PieceOfFurniture

Individual: PlaceMat1                               # Individuals in action descriptions are skolem
  Types: PlaceMat                                   # constants that are unified with real objects
                                                    # at runtime
[...]
```

The upper part describes the task *SetATable* as a subclass of *Action* with a set of *subActions*. The lower part consists of definitions of task-specific subclasses of generic action classes. The class *PutPlaceMatInFrontOfChair*, as an example, is defined as a subclass of the generic *PuttingSomethingSomewhere* action from the KnowRob ontology, with the additional restriction that the *objectActedOn* needs to be a *PlaceMat*, and the *toLocation* has to fulfill the requirements described for the class *Place1*, which by itself is described as some *Place* which is *inFrontOf-Generally* of some *Chair-PieceOfFurniture*. These structures allow to construct task descriptions at an arbitrary level of abstraction or specificity. The separation of task-specific classes in a task description and generic classes in the main action ontology allows to extract re-usable knowledge and put it into the main knowledge base, from where it can be inherited by other tasks derived from the same action classes.

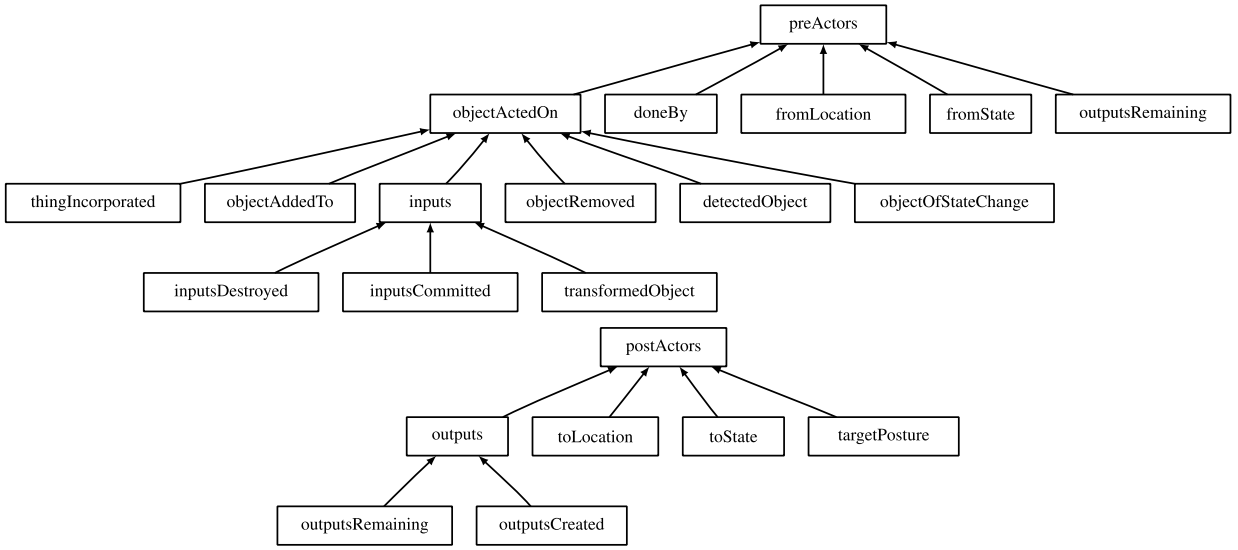

Fig. 10. Hierarchy of action-related properties. The upper part lists specializations of *preActors*, which describe the inputs of an action and the situation at its beginning. The *postActors* describe the outputs and post-conditions.

5.4. Effects of actions on objects

A robot has to reason about the effects of its actions on the objects it manipulates. Pick-and-place tasks merely change their positions (which can be described using the techniques introduced in Section 4.3. More complex activities like cooking meals involve more substantial changes of the objects themselves: Objects can be created or destroyed (e.g. when chopping vegetables) and can substantially change their types, appearance, and aggregate states (e.g. mixing and baking cookie dough). Robots therefore have to be able to represent which effects an action had and which objects got transformed into which other ones. In KnowRob, the representation of action effects consists of (1) *declarative specifications* of the inputs and outputs of an action, which can be used for searching for actions with the desired effects, and (2) *projection rules* to envision the world state after an action has been performed.

5.4.1. Declarative models of action effects

The declarative effect specifications extend the class taxonomy with information on the relation between actions and the objects they interact with. They are mainly used for searching for actions that have the desired effects and whose pre-conditions can be made true. Fig. 10 lists the most important properties that are used for representing action effects: *preActors*, displayed in the upper part of the figure, describe action properties that are supposed to hold before the action takes place. They include the agent (*doneBy*), the initial locations and states (*fromLocation*, *fromState*), or the *thingIncorporated* that is merged into the *objectAddedTo*. The *postActors* relations describe the outcome of an action, for example the *outputsCreated*, the *targetPosture* or 1 the *toLocation* of a transport action. These properties are used in class restrictions for describing the inputs, outputs, pre- and postconditions of an action class. Using common OWL inference, the robot can thus find actions with the desired properties while generalizing along the class taxonomy. For example, the robot can search for an *Action* that turns a *Device* from *DeviceStateOff* to *DeviceStateOn*, and will obtain the action class *TurningOnPoweredDevice*:

```
Class: TurningOnPoweredDevice
  SubClassOf:
      ControllingAPhysicalDevice
      objectOfStateChange some PhysicalDevice
      fromState value DeviceStateOff
      toState   value DeviceStateOn
      [...]
```

5.4.2. Projecting action effects

Complementary to the declarative effect axioms from the previous section, there are procedural projection rules for computing how a concrete world state would change after performing an action. While the declarative axioms are defined at the class level, the projection rules assert these relations for a concrete set of action and object instances. The consistency of the two representations has to be ensured by the programmer.

In the context of KnowRob, we are mainly interested in a light-weight and rather superficial form of envisioning that allows to spot major gaps in the task instructions. In the current implementation, the rules operate on a rather coarse, qualitative level and their predictions are neither very exact nor complete. A detailed prediction of the world state after an action has been performed would require much more sophisticated (and computationally expensive) methods that are for example based on physical simulation [20] – though even those are limited regarding the level of realism they can achieve.

The projection rules have been implemented as Prolog predicates that establish the links between an action instance and the objects that act as its inputs or outputs, are newly created or destroyed. They can be associated with actions at any level of the taxonomy, for example the rather generic rule for turning a device on, or a specific rule for dough that starts to bake when exposed to heat. Below is an example of a projection rule for the "mixing baking mix to batter" action. The predicate first checks its applicability conditions (the action is of the given type, the object is specified, and no projection of this action has been done so far). Then it creates the new object instances and asserts the relations between the input objects, the action, and the generated outputs. These relations correspond directly to the declarative specifications in the OWL ontology.

```
% Mixing baking mix to batter
project_action_effects(Action) :-

  owl_individual_of(Action, kr:'Mixing'),                    % Action of proper type
  \+ owl_has(Action, kr:outputsCreated, _),                  % No 'outputsCreated' property set

  owl_has(Action, kr:objectActedOn, Mix),                    % At least one objectActedOn is
  owl_individual_of(Mix, kr:'MixForBakedGoods'),             % of type MixForBakedGoods

  findall(Obj, owl_has(Action, kr:objectActedOn, Obj), Objs),  % Read all objects related by
                                                             % sub-properties of objectActedOn

  owl_instance_from_class(kr:'Batter',  Batter),             % Create instances of batter and
  owl_assert(Action, kr:objectAddedTo,  Batter),             % set it as objectAddedToand
  owl_assert(Action, kr:outputsCreated, Batter),             % outputsCreated of this action

  findall(O, (member(O, Objs),                               % The arguments of the mixing actions
              owl_assert(Action, kr:thingIncorporated, O)),_).  % have been incorporated into the batter
```

Initially, i.e. before the projection rule was applied, the only relation set for the action was the generic *objectActedOn* that linked it to an instance of *MixForBakedGoods*. Such generic relations can often be obtained from instructions. The projection rule then refined and extended the description by asserting properties such as *objectAddedTo*, *outputsCreated* and *thingIncorporated* that describe the relations in much more detail. This additional information is important for reasoning about how objects are transformed during a task and for identifying inconsistencies and knowledge gaps.

6. Evaluation

We characterize the performance of the proposed knowledge processing system along five dimensions: The size of the ontology, its runtime performance, examples of supported queries that cover different knowledge areas, examples of robot experiments in which KnowRob has been used, and the adoption of the open-source software library by the community, which we consider both as an indicator of its usability and its fitness for applications beyond those it was initially conceived for.

6.1. Knowledge content

The KnowRob ontology and its sub-ontologies currently include about 8000 classes describing events and temporal things, actions, objects and spatial things, as well as mathematical concepts and meta-information. There are about 130 action classes, 7000 object classes and 150 robot-specific concepts which can be described by over 300 kinds of properties. The classes cover a wide range of concepts about mobile manipulation robots, household chores, cooking and fetch-and-carry tasks. Extensions by third-party developers include knowledge about industrial assembly tasks, search and rescue in alpine environments, and underwater robotics.

6.2. Run time and scalability

Like in any Prolog program, the run time of a query depends on factors such as the size of the knowledge base, the order of predicates in a conjunction and their respective branching factors. Since KnowRob includes many external, i.e. non-Prolog, reasoners that may trigger complex computations, the runtime is also affected by which reasoners are to be considered. While general statements about the runtime efficiency are therefore difficult to make, the performance of a specific query can be predicted and tested very well. This aligns well with how KnowRob is used by the robot's control

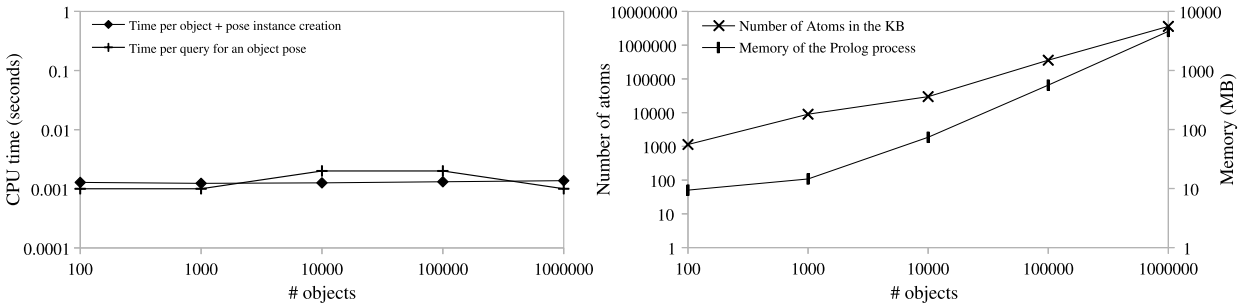

Fig. 11. Evaluation of the required CPU time (left) and memory (right) for creating large numbers of object perceptions in the knowledge base.

program: Control decisions are formulated as queries to the knowledge base that read required parameters. While their results are computed at runtime, the structure of the queries is known at plan design time, and their performance can thus be tested and optimized as needed.

To quantify the performance in a common task, we have simulated the creation of a large number of object perceptions as described in Section 4.3, which are among the most complex structures in KnowRob. For each simulated perception event, the system has to create the *VisualPerception* instance, the object instance and the pose matrix with its 16 elements (stored as OWL datatype properties). The following code has been used to create these structures for *N* ranging from *100* to *1,000,000* objects – which corresponds to storing one object perception per seconds for more than eleven days in a row. The *time()* predicate measures the CPU time consumed by evaluating a predicate. After creating the *N* object perceptions, we measure the time for querying for the pose of an arbitrary object.

```
create_objs(0).
create_objs(N) :-
  create_object_perception(kr:'Cup', [0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,0], ['VisualPerception'], _).
  N1 is N - 1,
  create_objs(N1).

# Create 100 object perceptions:
?- time(create_objs(100)).

# Query for object pose:
?- time(owl_individual_of(Obj, kr:'Cup'), current_object_pose(Obj, Pose)).
```

Fig. 11 (left) plots the average CPU time needed for creating one object-perception-pose structure and the time for querying for a pose after the *N* poses have been created. Both times are almost constant, the time for querying for the pose value jumps between one and two milliseconds, which is the resolution of the *time()* predicate. The right diagram plots the number of atoms in the knowledge base and the memory consumption of the Prolog UNIX process. Both increase almost linearly with the number of objects. Other queries that do not involve interaction with external components are usually answered in about 30–50 ms, queries that involve the on-demand analysis of CAD models (as an example of an expensive external inference procedure) require between 5–60 s, depending on the complexity of the object mesh.

6.3. *Example queries supported by* KnowRob

In this section, we will explain how the different representations contribute to the overall reasoning capabilities of KnowRob. In particular, we emphasize the benefit of having a common representation language that supports a wide range of queries which combine different kinds of information from different sources.

Reasoning about robot capabilities When processing instructions for a novel task, one cannot assume that all required capabilities are available on the robot. The system should therefore be able to decide whether important components or capabilities are missing by comparing the dependencies of a task description with a model of the available capabilities. While instructions usually do not come with suitable dependency specifications, they can often be inherited from the parent classes in the action ontology that the actions in the task are derived from (as explained in Section 5.3). The following example queries ask for the components and capabilities that are required for a task *MakingPancakes*. The returned dependencies have been inherited from the classes *Reaching* and *PickingUpAnObject* that some sub-actions of the *MakingPancakes* task are derived from. The two lower queries compare the required components and capabilities with those available on a given robot to identify missing ones. These examples show how robot models, task descriptions and the action ontology can be combined in a query.

```
?- required_comp_for_action(kr:'MakingPancakes', M).
M = srdl2comp:'ArmMotionController' ;
M = srdl2comp:'ArmComponent' ;

?- required_cap_for_action(kr:'MakingPancakes', M).
M = srdl2cap:'PickingUpAnObjectCapability' ;
M = srdl2cap:'ArmMotionCapability' ;

?- missing_comp_for_action(kr:'MakingPancakes', pr2:'PR2Robot1', M).
false.

?- missing_cap_for_action(kr:'MakingPancakes', pr2:'PR2Robot1', M).
M = srdl2cap:'PickingUpAnObjectCapability'
```

Exchanging information with other robots As long as all hardware components are available, the robot could try to launch software components that provide missing capabilities. Missing software components, e.g. detailed instructions for sub-tasks, environment maps or object models can also be downloaded from web-based knowledge bases such as RoboEarth [51]. For formulating queries to this system and for integrating the results with the existing knowledge base, it is important to have all information in the same language. The representation language used in RoboEarth [45] is a subset of the representations described in this paper, so information described in that language can simply be loaded into KnowRob. The following query downloads an environment map and an object model; it is formulated using the part-of relation between the building, floor and room (Section 4.2). After download, the system automatically downloads models for all object classes whose instances appear in the map if they are not available yet.

```
?- re_request_map_for([['kr:roomNumber', 3001],
                       ['kr:floorNumber', '3'],
                       ['kr:streetNumber', '45'],
                       ['rdfs:label', 'Karlstrasse']], M).

% Parsed "map.ks_3001" in 0.01 sec; 211 triples
* Missing object models:
  * Model for Cup
  % Parsed "cup.darkgray_cup.owl" in 0.01 sec; 12 triples

M = roboearth:'map.ks_3001' .
```

Locating objects in the environment To ground the abstract object descriptions in an instruction into actual objects in the environment, the robot needs to add actions for searching for these objects and for retrieving them from their storage locations. If the objects' locations are known, they can simply be looked up in the environment map (Section 4.2), but often this is not the case. Then, the system has to reason about likely locations given the available knowledge about the locations of other objects and their properties. This can for example be done using generic rules for storage locations [3] or based on a semantic similarity to other objects in the environment as explained in Section 4.1. If the inferred location is inside a container, the robot can query the semantic map for an articulation model that can parameterize the action for opening the container. The following query is an example how to obtain the opening trajectory of the container that is inferred to be the most likely storage location for milk; its result is show in Fig. 12 (left).

```
?- storagePlaceFor(StPlace, kr:'CowsMilk-Product'),
   owl_has(StPlace, kr:openingTrajectory, Traj),
   findall(P, (owl_has(Traj, kr:pointOnTrajectory, P) ), Traj).
```

Integrating experiences and observations of human actions Observations of human actions and memorized experiences can provide valuable information about objects, motions, locations and other action parameters. In our system, these kinds of memories are stored as instances of the respective action classes as explained in Section 5.2. This way, retrieving examples of the execution of an action reduces to reading all instances of the respective action class. An example query for the motion of taking a dinner plate out of a cupboard is given below, its results are visualized in Fig. 12 (right). This query combines information from the action instances with the action and object ontology and the logged continuous movements.

```
?- owl_individual_of(A, kr:'TakingSomething'),
   owl_triple(A, kr:objectActedOn, Obj),
   owl_individual_of(Obj, kr:'DinnerPlate'),
   owl_triple(A, kr:'trajectory-Arm', Tr),
   owl_triple(Tr, kr:pointOnArmTrajectory, P).
```

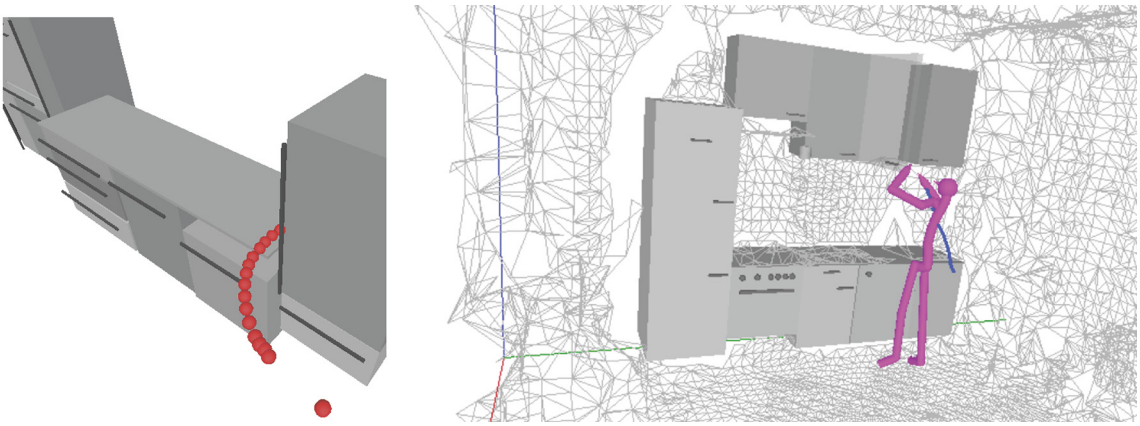

Fig. 12. Left: Opening trajectory of the refrigerator as result of a query for how to open the most likely storage place for milk. Right: Trajectory for taking a plate out of the cupboard selected from observations of a human performing a table-setting task.

Selecting functional object parts Using the part-based object representation, the system can automatically identify and locate relevant functional object parts in CAD models. Once the object is recognized in a scene, these relative locations can be translated into scene-global coordinates that the robot can use for interacting with the object. Below are example queries that are needed for selecting the right parts for pouring pancake batter onto a pancake maker. For picking up the bottle of pancake mix, the robot can query for a handle that is part of the bottle instance using the following query:

```
?- owl_triple(map:mix1, kr:properPhysicalParts, Part),
   annotation_handle(Part, kr:'Handle').
Part = kr:'Cone_wtOvROry' ;
```

For controlling the pouring motion, the part of the bottle that needs to be controlled is the opening. A reasonably general heuristic is to assume that the cone (as generalized cylinder) that is at the top of a bottle is its cap. This heuristic rule can be formulated as follows based on the geometric primitives identified in the CAD model by selecting all cones and sorting them based on their z coordinate. This example shows how logical rules can be formulated in the knowledge base that are evaluated on the object components extracted using geometric analysis.

```
bottle_cap(Obj, Cap) :-
    findall(Z-P,
      (rdf_triple(kr:properPhysicalParts, Obj, P),
       owl_individual_of(P, kr:'Cone'),
       objpart_pos(P, [_,_,Z])), ConePos),
    keysort(ConePos, ConePosAsc),
    last(ConePosAsc, _-Cap).

?- bottle_cap(kr:'pancakemix1', Cap).
Cap = kr:'Cone_vcRxyUbK'.
```

6.4. Example use cases on a physical robot

The KnowRob system has been applied in several experiments on physical robots. Fig. 13 shows examples of three tasks on two different robots: The left two pictures show a drink serving task performed by a PR2 robot (left) and an Amigo robot (center). Both robots execute the same task description (Section 5.3) and parameterize it with semantic models of the environment (Section 4.2), of the drink pack to be served (Section 4.1), and the articulated model of the cabinet that describes how the door can be opened (Section 4.5). The robots used abstract models of their hardware and their capabilities (Section 4.6) to decide if they are able to perform the task.

The right image in Fig. 13 shows a pancake making task realized on the PR2 robot. In this task, KnowRob has again been used for representing the actions to be performed, the environment model and the objects to be manipulated. Especially the actions for pouring batter onto the pancake maker and for flipping a pancake include complex motions that have been parameterized with part-based object models that have automatically been derived from CAD models (Section 4.5). Based on these models, the robot could determine which parts of the objects it needs to interact with, e.g. to grasp the spatulas at their handles and to consider the position of the bottle opening when pouring.

Fig. 13. Example applications of KnowRob on physical robots: The PR2 (left) and the Amigo robot (center) serve a drink from a cupboard in different environments based on semantic models from KnowRob. The pancake making task (right) as well as the involved objects have also been modeled in KnowRob and executed using the CRAM plan-based controllers on the PR2 robot.

6.5. Open-source release & community adoption

KnowRob has been available as open-source software for several years, and several research groups and collaborative research projects have chosen it as knowledge base for their robot applications. While no classical evaluation measure, the breadth of applications that third-party researchers use the system for are, in our opinion, an indication of the usability and usefulness of the system. For example, several European research projects use KnowRob for representing knowledge to be exchanged between robots (RoboEarth[5]), for integrating information from the Web with task demonstrations given by humans (RoboHow[6]), for elderly-care robots (SRS[7]), for assembly tasks in industrial applications (SMErobotics[8]), for reasoning about safe human–robot cooperation (SAPHARI[9]), and for multi-robot search-and-rescue tasks in alpine disaster scenarios (SHERPA[10]). Other (not yet published) applications include underwater robotics and multi-player online games that use the spatio-temporal object representation for virtual scenes.

To further promote the use of knowledge representation and reasoning methods for robotics applications, we have created a web-based version of the KnowRob system called Open-EASE. Its core is formed by the same KnowRob knowledge base that also runs on the robot, but users (or robots) can access and query the knowledge base over the Internet using a WebSocket interface.[11] Open-EASE does not require any installation and offers a rich set of browser-based visualizations for query results.

7. Discussion and related work

In this paper, we have discussed how we can structure a knowledge base for robots to help them with their tasks. Our main insight is that we need a combination of different knowledge areas, different knowledge sources and different inference mechanisms to cover the breadth and depth of required knowledge and inferences. At the same time, all components need to be integrated in a coherent knowledge base such that the robot can combine different knowledge sources and inference methods in a single query and integrate their results. We therefore use a shared ontology as common representation and Prolog as interlingua for integrating inference techniques.

Finding appropriate representations for a robot's knowledge base has been a research topic for decades, dating back to the seminal work on the Shakey robot [31] that already used an internal world model based on predicate logics. Compared to Shakey, modern robots have perception techniques that can handle the complexity of real scenes and manipulation capabilities that allow them to interact with objects. Both aspects lead to significantly more complex scene representations and to a massive increase in knowledge about actions and objects that is required for performing actions.

While various methods have been developed in the AI community for representing and reasoning about temporal relations, action effects and changing situations, most of them focus on individual inference problems: Allen's interval calculus [1], for example, is mainly used for reasoning about temporal intervals, the Region Connection Calculus [34] extends it to two-dimensional (spatial) problems. The Situation Calculus [25,35] and the related Fluent calculus [47] focus on the representation and reasoning about changing domains, e.g. caused by robot actions. The Qualitative Process Theory by Forbus [13] allows qualitative inference about physical or chemical processes. Planning languages like STRIPS [12], PDDL [16], or Hierar-

[5] http://www.roboearth.org.
[6] http://www.robohow.eu.
[7] http://www.srs-project.eu.
[8] http://www.smerobotics.org/.
[9] http://saphari.eu/.
[10] http://sherpa-project.eu/.
[11] http://www.open-ease.org.

chical Task Networks [9] specialize in the generation of plans for achieving a given goal. Other rather formalized approaches for representing a robot's knowledge [48] allow reasoning about what the robot knows and what it does not know, but lack support for e.g. temporal reasoning, detailed spatial representations, information about object types or about processes in the environment.

Similar to reasoning methods, there are also many knowledge bases that partially cover a robot's needs: General-purpose knowledge bases like Cyc [23] or SUMO [30] provide a large breadth of encyclopedic knowledge, but often lack the depth in topis like object manipulation that is required by robots. Recent efforts to automate the construction of knowledge bases by extracting encyclopedic knowledge from sources like Wikipedia [55,40,17] can provide knowledge in specialized areas, though often not directly relevant for robotics (e.g. countries, people or historic events).

On the other end, representations developed in robotics have largely focused on modeling and reasoning with uncertainty, leading to the development of many sophisticated, though often special-purpose probabilistic models [49]. However, most of these models are specialized for a single modality, for example perception [19], articulation model learning [39], or robot localization [8], and usually lack clear semantics. Just recently, research in semantic environment maps has started to investigate models that combine spatial with grounded semantic representations [15,58]. Wyatt et al. present a system that integrates geometric and conceptual spatial representations with planning and learning techniques [57]. The system is able to reason about knowledge gaps and tries to resolve them using autonomous learning, though not for object manipulation tasks.

In our research, we try to combine (parts of) these approaches in a common system to provide robots with comprehensive knowledge and inference capabilities. The representation of object poses, for instance, is similar to the Fluent calculus [47], but also stores the provenance of information, i.e. if the robot believes, predicts or desires an object to be at some location. This allows further interpretation of the information as well as describing different (possible) world states without causing conflicts in the knowledge base, which is important if the belief results from noisy perception. The action representation as hierarchical partially-ordered plans with prerequisites, effects, and temporal information is related to Hierarchical Task Networks [9], but extended with qualitative projection methods for processes that are started as side-effects of actions, which are inspired by the Qualitative Process Theory Forbus [13]. Inference about temporal relations between actions or events is implemented according to Allen's interval calculus [1].

All inferences can be combined in Prolog queries, read the required information from the knowledge stored as OWL statements, and return their results in terms of OWL descriptions. The knowledge base can be populated automatically with perception results and log data of robot actions. Procedural attachments allow symbolic inference about sub-symbolic data by computing semantic information from it on demand at query time. With these capabilities, KnowRob is significantly more expressive and flexible than other robot knowledge bases that purely rely on the rather limited OWL inference, for example ORO [22] or OUR-K [24].

Acknowledgements

This work is supported in part by the EU FP7 Projects *RoboEarth* (grant number 248942), *RoboHow* (grant number 288533), and *ACat* (grant number 600578).

References

[1] J. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.
[2] R. Arnaud, M. Barnes, COLLADA: Sailing the Gulf of 3D Digital Content Creation, AK Peters, Ltd., 2006.
[3] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, M. Tenorth, Robotic roommates making pancakes, in: 11th IEEE–RAS International Conference on Humanoid Robots, Bled, Slovenia, 2011.
[4] M. Beetz, M. Tenorth, D. Jain, J. Bandouch, Towards automated models of activities of daily life, Technol. Disabil. 22 (1–2) (2010) 27–40.
[5] M. Beetz, M. Tenorth, J. Winkler, Open-EASE – a knowledge processing service for robots and robotics/AI researchers, in: IEEE International Conference on Robotics and Automation, ICRA, Seattle, Washington, USA, 2015, accepted for publication.
[6] R.A. Brooks, Elephants don't play chess, Robot. Auton. Syst. 6 (1990) 3–15.
[7] E. Davis, A logical framework for solid object physics, Technical report 245, 1986.
[8] H. Durrant-Whyte, T. Bailey, Simultaneous localization and mapping: part 1, IEEE Robot. Autom. Mag. 13 (2) (June 2006) 99–110.
[9] K. Erol, J. Hendler, D. Nau, HTN planning: complexity and expressivity, in: Proceedings of the National Conference on Artificial Intelligence, John Wiley & Sons LTD., 1994, pp. 1123–1128.
[10] C. Fellbaum, WordNet: An Electronic Lexical Database, MIT Press, USA, 1998.
[11] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A.A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, C. Welty, Building Watson: an overview of the DeepQA project, AI Mag. 31 (3) (2010) 59–79.
[12] R.O. Fikes, N.J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Technical report 43R, AI Center, SRI International, 1971.
[13] K. Forbus, Qualitative process theory, Artif. Intell. 24 (1984) 85–168.
[14] M. Fox, D. Long, PDDL2.1: an extension of PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (2003) 61–124.
[15] C. Galindo, J.-A. Fernández-Madrigal, J. González, A. Saffiotti, Robot task planning using semantic maps, Robot. Auton. Syst. 56 (11) (2008) 955–966.
[16] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL – the planning domain definition language, AIPS-98 planning committee, 1998.
[17] J. Hoffart, F. Suchanek, K. Berberich, G. Weikum, Yago2: a spatially and temporally enhanced knowledge base from Wikipedia, Artif. Intell. 194 (2013) 28–61.
[18] L.P. Kaelbling, M.L. Littman, A.R. Cassandra, Planning and acting in partially observable stochastic domains, Artif. Intell. 101 (1–2) (1998) 99–134.
[19] D. Kragic, M. Vincze, Vision for robotics, Found. Trends Robot. 1 (1) (2009) 1–78.

[20] L. Kunze, M.E. Dolha, E. Guzman, M. Beetz, Simulation-based temporal projection of everyday robot object manipulation, in: Yolum, Tumer, Stone, Sonenberg (Eds.), Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2011, Taipei, Taiwan, IFAAMAS, 2011.
[21] L. Kunze, T. Roehm, M. Beetz, Towards semantic robot description languages, in: IEEE International Conference on Robotics and Automation, ICRA, Shanghai, China, 2011, pp. 5589–5595.
[22] S. Lemaignan, Grounding the interaction: knowledge management for interactive robots, PhD thesis, CNRS, Laboratoire d'Analyse et d'Architecture des Systèmes, Technische Universität München, Intelligent Autonomous Systems lab, 2012.
[23] D. Lenat, CYC: a large-scale investment in knowledge infrastructure, Commun. ACM 38 (11) (1995) 33–38.
[24] G.H. Lim, I.H. Suh, H. Suh, Ontology-based unified robot knowledge for service robots in indoor environments, IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 41 (3) (2011) 492–509.
[25] J. McCarthy, Situations, actions and causal laws, Technical report, Stanford University, 1963, reprinted in: M. Minsky (Ed.), Semantic Information Processing, MIT Press, 1968.
[26] D. McDermott, Robot planning, AI Mag. 13 (2) (1992) 55–79.
[27] D. McDermott, The formal semantics of processes in PDDL, in: Proceedings of the ICAPS Workshop on PDDL, 2003.
[28] L. Morgenstern, Mid-sized axiomatizations of commonsense problems: a case study in egg cracking, Stud. Log. 67 (3) (2001) 333–384.
[29] L. Mösenlechner, M. Beetz, Parameterizing actions to have the appropriate effects, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, San Francisco, CA, USA, 2011.
[30] I. Niles, A. Pease, Towards a standard upper ontology, in: Proceedings of the International Conference on Formal Ontology in Information Systems, vol. 2001, ACM, 2001, pp. 2–9.
[31] N.J. Nilsson, Shakey the robot, Technical report 323, AI Center, SRI International, Menlo Park, CA, USA, 1984.
[32] D. Pangercic, M. Tenorth, D. Jain, M. Beetz, Combining perception and knowledge processing for everyday manipulation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Taipei, Taiwan, 2010, pp. 1065–1071.
[33] D. Pangercic, M. Tenorth, B. Pitzer, M. Beetz, Semantic object maps for robotic housework – representation, acquisition and use, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vilamoura, Portugal, 2012.
[34] D.A. Randell, Z. Cui, A.G. Cohn, A spatial logic based on regions and connection, in: Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning, 1992.
[35] R. Reiter, The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression, in: Artificial Intelligence and Mathematical Theory of Computation, Academic Press Professional, Inc., 1991, pp. 359–380.
[36] R. Reiter, Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press, 2001.
[37] M. Schuster, D. Jain, M. Tenorth, M. Beetz, Learning organizational principles in human environments, in: IEEE International Conference on Robotics and Automation, ICRA, St. Paul, MN, USA, 2012.
[38] B. Siciliano, O. Khatib (Eds.), Springer Handbook of Robotics, Springer, Berlin, Heidelberg, 2008.
[39] J. Sturm, C. Stachniss, W. Burgard, Learning kinematic models for articulated objects, J. Artif. Intell. Res. (2011).
[40] F. Suchanek, G. Kasneci, G. Weikum, Yago: a core of semantic knowledge, in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 697–706.
[41] M. Tenorth, M. Beetz, Exchange of action-related information among autonomous robots, in: 12th International Conference on Intelligent Autonomous Systems, 2012.
[42] M. Tenorth, M. Beetz, A unified representation for reasoning about robot actions, processes, and their effects on objects, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vilamoura, Portugal, 2012.
[43] M. Tenorth, M. Beetz, KnowRob – a knowledge processing infrastructure for cognition-enabled robots, Int. J. Robot. Res. 32 (5) (2013) 566–590.
[44] M. Tenorth, U. Klank, D. Pangercic, M. Beetz, Web-enabled robots – robots that use the Web as an information resource, IEEE Robot. Autom. Mag. 18 (2) (2011) 58–68.
[45] M. Tenorth, A.C. Perzylo, R. Lafrenz, M. Beetz, Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth framework, IEEE Trans. Autom. Sci. Eng. 10 (3) (2013) 643–651, best paper award finalist.
[46] M. Tenorth, S. Profanter, F. Balint-Benczedi, M. Beetz, Decomposing CAD models of objects of daily use and reasoning about their functional parts, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Tokyo Big Sight, Japan, 2013, pp. 5943–5949.
[47] M. Thielscher, Introduction to the fluent calculus, Electron. Trans. Artif. Intell. 2 (1998) 179–192.
[48] M. Thielscher, Representing the knowledge of a robot, in: International Conference on Principles of Knowledge Representation and Reasoning, Breckenridge, CO, USA, 2000, pp. 109–120.
[49] S. Thrun, W. Burgard, D. Fox, Probabilistic Robotics, MIT Press, Cambridge, 2005.
[50] W3C, OWL 2 Web ontology language: structural specification and functional-style syntax, World Wide Web Consortium, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027, 2009.
[51] M. Waibel, M. Beetz, R. D'Andrea, R. Janssen, M. Tenorth, J. Civera, J. Elfring, D. Gálvez-López, K. Häussermann, J. Montiel, A. Perzylo, B. Schießle, O. Zweigle, R. van de Molengraft, RoboEarth – a world wide web for robots, IEEE Robot. Autom. Mag. 18 (2) (2011) 69–82.
[52] J. Wielemaker, G. Schreiber, B. Wielinga, Prolog-based infrastructure for RDF: performance and scalability, in: D. Fensel, K. Sycara, J. Mylopoulos (Eds.), The Semantic Web – Proceedings, ISWC'03, Sanibel Island, Florida, in: LNCS, vol. 2870, Springer Verlag, Berlin, Germany, 2003, pp. 644–658.
[53] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, SWI-Prolog, Theory Pract. Log. Program. (2012) 67–96.
[54] J. Winkler, M. Tenorth, A.K. Bozcuoglu, M. Beetz, CRAMm – memories for robots performing everyday manipulation activities, Adv. Cogn. Syst. 3 (2014) 47–66.
[55] F. Wu, D.S. Weld, Autonomously semantifying Wikipedia, in: CIKM '07: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2007, pp. 41–50.
[56] Z. Wu, M. Palmer, Verbs semantics and lexical selection, in: Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, Morristown, NJ, USA, 1994, pp. 133–138.
[57] J.L. Wyatt, A. Aydemir, M. Brenner, M. Hanheide, N. Hawes, P. Jensfelt, M. Kristan, G.-J.M. Kruijff, P. Lison, A. Pronobis, K. Sjöö, D. Skočaj, A. Vrečko, H. Zender, M. Zillich, Self-understanding and self-extension: a systems and representational approach, IEEE Trans. Auton. Ment. Dev. 2 (4) (2010) 282–303.
[58] H. Zender, O. Martínez Mozos, P. Jensfelt, G. Kruijff, W. Burgard, Conceptual spatial representations for indoor mobile robots, Robot. Auton. Syst. 56 (6) (2008) 493–502.