KNOWROB – Knowledge processing for autonomous personal robots

Conference Paper · November 2009

DOI: 10.1109/IROS.2009.5354602 · Source: IEEE Xplore

2 authors:

Moritz Tenorth

50 PUBLICATIONS **2,022** CITATIONS

SEE PROFILE

Michael Beetz

Universität Bremen

436 PUBLICATIONS **9,787** CITATIONS

SEE PROFILE

KNOWROB — Knowledge Processing for Autonomous Personal Robots

Moritz Tenorth and Michael Beetz
Intelligent Autonomous Systems, Technische Universität München
{tenorth, beetz}@cs.tum.edu

***Abstract*— Knowledge processing is an essential technique for enabling autonomous robots to do the *right thing* to the *right object* in the *right way*. Using knowledge processing the robots can achieve more flexible and general behavior and better performance. While knowledge representation and reasoning has been a well-established research field in Artificial Intelligence for several decades, little work has been done to design and realize knowledge processing mechanisms for the use in the context of robotic control.**

In this paper, we report on KNOWROB, a knowledge processing system particularly designed for autonomous personal robots. KNOWROB is a first-order knowledge representation based on description logics that provides specific mechanisms and tools for action-centered representation, for the automated acquisition of grounded concepts through observation and experience, for reasoning about and managing uncertainty, and for fast inference — knowledge processing features that are particularly necessary for autonomous robot control.

I. INTRODUCTION

As the complexity of tasks to be accomplished by personal robots is steadily increasing we cannot afford to explicitly spell out every detail of every course of action in every conceivable situation.

Consider, for example, a personal robot that is to set the table. Obviously, we cannot specify all aspects of the required actions explicitly: where to stand when reaching for a cup, which hand to take, which grasp to use, etc — and these aspects for every object and every context. Rather, we have to specify *general* and *flexible* control routines that automatically adjust their execution to the particular situation at hand.

In order to code control routines in such general manners, the robot has to be capable of *inferring* the detailed course of action and the action parameterizations using the required *abstract and symbolic knowledge* pieces. That is, knowledge processing is a resource for doing the *right thing* to the *right object* in the *right way*. To this end, robots must be equipped with a comprehensive body of knowledge and dedicated knowledge processing capabilities that allow for better state estimation, context assessment, and better informed action selection and parameterization.

To be applicable to autonomous robot control, knowledge representation and processing should address the following aspects, which are typically not sufficiently covered by AI knowledge representation and processing mechanisms.

- *Action-centered knowledge representation.* Action-related concepts like the places where certain manipulations can be performed or the grasp of a bottle for filling a glass are central for a robot performing manipulation tasks.
- *Automated acquisition of grounded concepts through observation and experience.* Symbolic knowledge in the robot's knowledge base has to be grounded in data structures from the perception system and in parametrizations of actions. The knowledge processing system should thus work directly on the data structures used for robot control such as 3D environment models, or log data from plan execution, and should abstract the high-dimensional data into compact symbolic representations for efficient reasoning (Figure 1).
- *Managing uncertainty.* Uncertainty caused by sensor noise, by limited observability, by hallucinated object detections, by incomplete knowledge, and by unreliable and inaccurate actions must be taken into account.
- *Fast inference.* The knowledge processing system of a robot must provide answers quickly compared to the execution times of manipulation tasks.

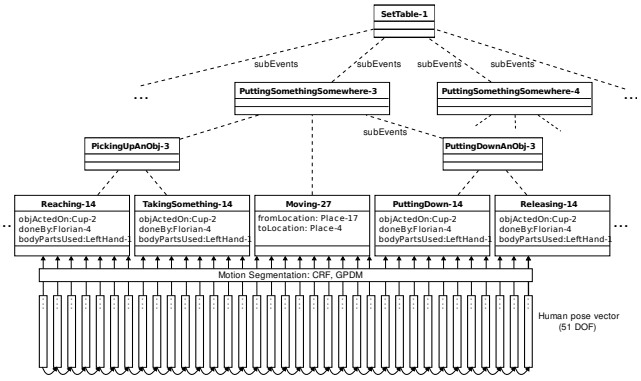

Fig. 1. Hierarchical action model: The continuous pose sequence (bottom) is segmented into primitive movements (Reaching, etc) and combined with additional information from the sensor network and the object recognition system. The model is automatically constructed by matching definitions of higher-level concepts against these observations, e.g. that picking something up subsumes reaching towards and taking it.

In this paper we describe key components and mechanisms for a practical knowledge processing system specifically designed for autonomous robots that are to perform everyday manipulation tasks. The main contributions of this paper are:

- We describe a complete, practical knowledge processing system that integrates encyclopedic knowledge, an environment model, action-based reasoning, and human observations and allows to access all this information in a uniform, symbolic way.

- The system allows for symbolic queries of continuous sensor data observed in a real-world environment.
- We introduce *computable* classes and properties for creating instances from observed data and *action models* as a powerful means for discovering a class structure among action-related concepts.

In the remainder of this paper we proceed as follows. We start with an overview over related work, explain the general structure of the system and continue with chapters about the knowledge representation, the inclusion of external data using computable classes and properties, and the learning of action models from data. We present the results we obtained using the presented methods and finish with our conclusions.

II. Related Work

Knowledge representation is a well-researched topic in AI, and there exist huge knowledge bases covering encyclopedic knowledge with large breadth and width, one of the most prominent being Cyc [1]. Unfortunately, they are only of limited use for autonomous robot control: As mentioned in the introduction, robots have very specific demands which are usually not met by these knowledge bases.

One of the main problems is that the abstract concepts therein are not linked to the perception and actuation system of the robot. This link between the abstract knowledge representation and a particular control system of a robot is called the *(symbol) grounding* problem [2]. There are few knowledge representation systems tailored to robotics applications. The *Grounded Situation Models*, described by Mavridis [3], integrate continuous and stochastic data with categorical information in a grounded fashion, but their "world" is just a white table with coloured blocks on top. A formal knowledge representation is missing, as is the complexity inherent in real-world tasks. The knowledge representation of the PEIS ecology project [4] focuses mainly on the representation of objects and the perceptual anchoring, i.e. maintaining the link between perceptual sensations and symbolic concepts, but does not deal with action-related knowledge. The probably most related system is the OMRKF framework by Hong Suh et al. [5], which is also a Prolog-based knowledge representation modeling objects and perceptual concepts as well as actions and situations. Our system differs from these systems in that it features a robot acting in a real human environment, observing and reasoning about human and robot activities, and combines these observations with encyclopedic and common-sense knowledge. Our system is based on state-of-the-art semantic web technology that allows us to re-use existing sources of knowledge whenever available.

III. Knowledge processing framework

We will explain the system using the an example of setting a table, a typical mobile manipulation task for a household robot.

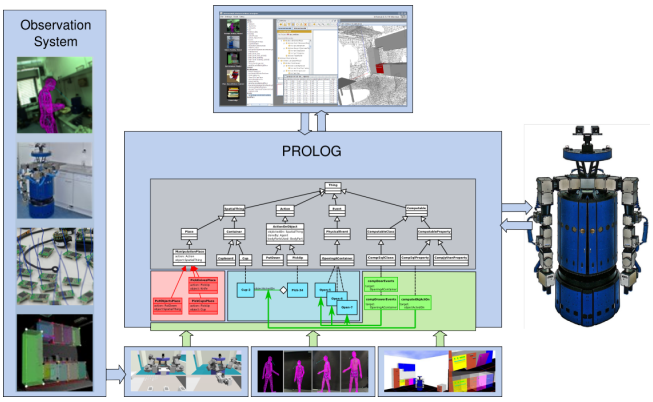

Fig. 2. System structure; the block in the center is explained in more detail in Figure 3. The data produced by the perception (robot log data, human motion tracking and environment information) can be accessed from within the knowledge base using computable predicates (displayed in green).

A. System Overview

Figure 2 shows the main modules of the system. Our robot is acting in a sensor-equipped kitchen environment [6] which includes laser range finders, cameras, magnetic sensors for detecting if a cupboard is opened, and radio-frequency identification (RFID) tag readers for identifying objects. Further perception modules create 3D environment maps, track human motions, and record log data of robot actitivies (Section III-C). These data structures are linked to the knowledge processing system using *computable* classes and properties, which load the data and represent them as instances in the knowledge base (Section III-D).

Our implementation is based on SWI Prolog and its Semantic Web library which serves for loading and accessing ontologies represented in the Web Ontology Language (OWL) using Prolog predicates like *owl_assert()* or *owl_query()*. The knowledge base can either be queried from within the robot control program, from the command line or using a graphical user interface. An interface to the YARP [7] middleware faciliates the integration into the robot control program. The graphical user interface allows to manually analyze the data, to send and to visualize queries; all visualisations shown in this paper were created using this tool.

In this paper, the term "autonomous system" does not only refer to the mobile robot, but the integrated hardware-software system which comprises mobile and fixed components like a robot and a sensor network; "autonomous" is used as "no human interaction during operation".

B. Knowledge Representation

Figure 3 provides an overview of the represented types of knowledge and the different modules of the system.

The knowledge is represented in description logics using OWL. In description logics, there are two main levels of modeling: *Classes*, sometimes also referred to as *concepts*, and *instances*. The class level contains abstract terminological knowledge like the types of objects, events and actions, organized in a taxonomic structure. Instances

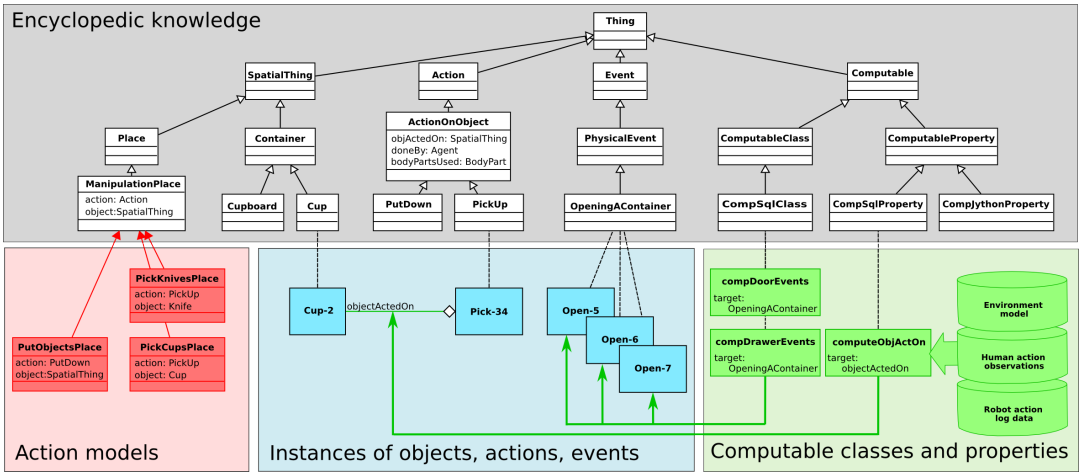

Fig. 3. Overview diagram showing the relations between encyclopedic class knowledge (upper grey block), instances of these classes (bottom center), computable classes and properties which load data into the system (bottom right), and action models which are learned from observations on demand (bottom left). Computables create instances from data, action models learn classes from a set of instances.

represent concrete, physical objects or actually performed actions. *Properties* link classes or instances. All relations are formulated as *(Subject, Property, Object)* triples.

The encyclopedic knowledge, depicted in the upper grey part, models classes of things in the environment and provides the general categories the robot describes his environment with. The overall structure and some of the concepts are inspired by the Cyc ontology [1]. We selected those concepts that are relevant for a mobile robot and added classes where the level of detail was not sufficient.

A few instances are shown in the blue block in the bottom center. In our system, they model either physical objects, performed actions or observed events. These instances can be created automatically from observations using *computable* classes and properties which are depicted in the green block in the bottom right and described in detail in Section III-D.

While computables serve for creating *instances* from observations, action models generate new *classes* from the observed data (depicted in the red block in the bottom left). Using embedded classifiers, the system searches for groups of instances that share common properties and could be put into the same subclass, thereby effectively creating a subclass of the current concept. In this paper, we use the example of learning subclasses of "ManipulationPlace". However, our approach is more general and can be used for any kind of action-related concept, i.e. each class that can be characterized by its role in actions. Action models are described in Section III-E.

C. Data structures from the Perception System

Data created from the different perception modules are the basis for many reasoning tasks. Information about fixed objects in the environment is provided by a semantic environment map created from 3D laser scans. During the mapping process, the point clouds created by the laser scanner are segmented into single objects, which are then classified into categories like "Cupboard" or "Dishwasher" as described in [8]. These categories correspond to classes in the knowledge base, the detected objects are thus represented as instances of these classes, together with properties like their dimensions or position.

Mobile objects are detected by the vision system [9] or a set of radio-frequency ID (RFID) tag readers in the kitchen. Therefore, the respective instances have a link to the vision model which can be used to match it in the camera image, or the ID of the tag attached to them. Binary sensors in the sensor network [10] like magnetic switches in doors and drawers directly produce events of a specified type like "OpeningADrawer".

Observations of human actions are obtained from a markerless full-body pose tracking system that estimates human motions in 51 degrees of freedom based on images from fixed cameras [11]. Robot actions are recorded using hybrid automata that are specified in the Robot Learning Language (RoLL, [12]). These accepting automata allow for recording detailed log files of performed tasks in an action-centric way, including internal and external events and the robot's belief about the world.

Links from action concepts in the knowledge representation to plans in the planning system tell the robot how to execute these actions. Properties of the action concept like the object to be manipulated become parameters of the plan, so that the robot handles a cup different from a plate.

D. Computable Classes and Properties

Interfacing the observation system and loading observations into the knowledge representation is one main purpose of computables. As depicted by the green arrows in Figure 3, computables create either instances or relations between instances as specified in their *target* property.

By linking computables to the respective properties using the *target* relation, we keep the representation of the knowledge itself separate from technical issues like the automatic creation of instances. Moreover, this setup allows to define several computables for one property (e.g. one that reads object information from a vision system and another one that uses RFID tag readers).

The definition of a computable property specifies the commands for reading the *object* or the *subject* in an OWL

triple:

```
owl_assert(computeObjX, type, SqlProperty)
owl_assert(computeObjX, target, objX)
owl_assert(computeObjX, selectObject,
    "SELECT objx FROM objects WHERE action = '?SUBJECT'")
owl_assert(computeObjX, selectSubject,
    "SELECT action FROM objects WHERE objx = '?OBJECT'")
```

For implementing computables, we modified the functions for accessing the knowledge base so that they include computables with a matching *target* into the reasoning process. The result of computable properties is thereby equivalent to instances that have been created manually in the knowledge representation.

In addition to loading observations into the system, computable properties can also be used for calculating new relations from the existing knowledge. For example, spatial relations between objects, like *on*, *in* or *below*, are completely determined by the positions of these objects. Storing both their positions and all possible relations would cause much overhead, calculating the relations on demand is much more elegant. For such applications, we are using *JythonComputables*, which execute a Jython script instead of sending an SQL query, but otherwise work like *SqlComputables*.

E. Action Model Learning

Action models seek for a class structure among a set of entities, for example all observed *ManipulationPlaces*, by grouping those that are similar with respect to certain properties. The goal is to discover subclasses like a “Put-down-objects-place” or a “Pick-up-cups-place”. Technically, this is done by learning a classifier on a set of observations, the resulting rules then describe the different classes.

The specification of an action model (called the *intensional* model) describes its general structure. It comprises a set of observable features and the class whose subclasses are to be found. This specification is independent of the environment and just depends on the relation to be modeled.

An environment-specific instance of the *intensional* model, an *extensional* model consisting of classifier rules, is learned on demand based on the available data. The *intensional* model is specified once by a human expert, all *extensional* models are learned autonomously from this specification. In the observed training data, both the observables and the class values are known, the process can thus be regarded as a kind of self-supervised learning.

The definition of an intensional action model is shown in the following listing and comprises a specification of the training set of actions (in this case all observed actions of type *ActionOnObject*), the *observable* features and the *predictable* class. The associated *extensional* model consists of a set of classifier rules mapping from (a subset of) the observables to the predictable class.

```
owl_assert(manipActions,   onProperty,   actionType)
owl_assert(manipActions,   hasValue,     'ActionOnObject')
owl_assert(manipActions,   type,         'Restriction')

owl_assert(manipPosModel,  type,         'ActionModel')
```

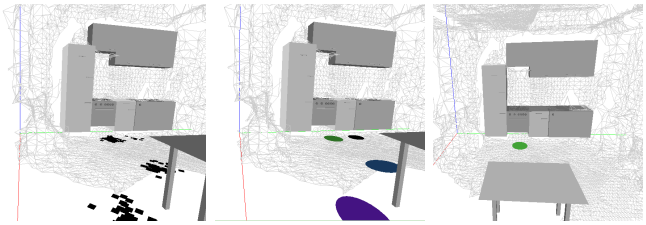

Fig. 4. Observed positions of manipulation actions (left), the positions clustered into places (center), and the result that has been learned as the ”place for picking up pieces of tableware” (right). The pictures are visualized results of queries to the knowledge processing system.

```
owl_assert(manipPosModel,  forAction,    manipActions)

owl_assert(manipPosModel,  observable,   actionType)
owl_assert(manipPosModel,  observable,   objectClass)
owl_assert(manipPosModel,  observable,   objX)
owl_assert(manipPosModel,  observable,   objY)
owl_assert(manipPosModel,  observable,   objZ)

owl_assert(manipPosModel,  predictable,  robotPlace)
```

When learning an extensional model, the first reasoning step is to load the training data into the knowledge processing system, usually via computable properties (Figure 4 left). Afterwards, the data are abstracted using data mining techniques; in particular, the position are aggregated to clusters with respect to the Euclidean distance. The resulting clusters are depicted as ellipses in Figure 4 (center) and each cluster is given a name: *P1* (green cluster), *P2* (black cluster),... Clusters of positions are abstractly represented as places.

actionType	objectClass	objX	objY	objZ	robotPlace
PickingUpAnObject	DinnerPlate	0.48	1.77	0.91	P1
PickingUpAnObject	TableKnife	0.32	2.04	0.90	P2
PickingUpAnObject	DinnerPlate	0.18	1.78	0.92	P1
PuttingDownAnObject	DinnerPlate	2.90	1.99	0.73	P3
PuttingDownAnObject	TableKnife	3.07	2.12	0.71	P3
PuttingDownAnObject	DinnerPlate	3.21	1.51	0.72	P4

TABLE I

EXCERPT OF DATA OBSERVED WHILE THE ROBOT SET A TABLE.

The system then reads the data of the observable properties in the training set (Table I) and uses decision tree learning to extract rules which map from the actions performed and the types and positions of objects to the place where the robot was standing. In this small example, the rules are:

$$actionType = PickingUpAnObject \wedge objectClass = DinnerPlate \longrightarrow P1\ (prob: 1.00)$$

$$actionType = PickingUpAnObject \wedge objectClass = TableKnife \longrightarrow P2\ (prob: 1.00)$$

$$actionType = PuttingDownAnObject \wedge objX < 3.15 \longrightarrow P3\ (prob: 1.00)$$

$$actionType = PuttingDownAnObject \wedge objX >= 3.15 \longrightarrow P4\ (prob: 1.00)$$

The decision tree learning algorithm further outputs a confidence value which describes the amount of training examples that support a given rule.This value can be useful in real settings where the classification is usually not as perfect as in this little example. While this value yields the confidence in a single inference step, the description logics-based framework does not further propagate this uncertainty. We are currently investigating the use of probabilistic logical models like Markov Logic Networks, but due to their limited

scalability, we will probably use them only for certain inference tasks that require full uncertainty propagation.

As can be seen from the rules above, the pick-up places are mainly determined by the object manipulated; objects of different kinds are stored at different places in a kitchen. The put-down places, in contrast, can best be distinguished based on the seating location, while the types of manipulated objects provide no clear separation in this case. These rules are equivalent to concept definitions in description logics and thus extend the class hierarchy:

$$PickPlatePlace \sqsubseteq actionType.PickingUpAnObject \sqcap objectClass.DinnerPlate$$

In the remainder of the paper, we will use a shorter notation for action models that just includes the sets of observable and predictable properties:

```
actionmodel(
  observables(actionType, objectClass, objX, objY, objZ)
  predictables(robotPlace, robotOrientation))
```

IV. Results

Measuring the performance of a knowledge processing system is difficult since many of its advantages, like greater flexibility or generality, are hard to capture in numbers. In our opinion, the two most important aspects are the variety of queries that become possible having such a system, and the speed at which the answers are generated.

A. Realtime Performance

We measured the time for different queries on a standard dual-core laptop running Debian Linux. The queries on the environment model presented in the next sections all take less than 10ms. Since much of the knowledge is directly available, not much inference is needed and the system achieves database-like performance. Learning an action model is more complex, as the training data has to be loaded from the database, the classifier has to be built and applied to the observation at hand. For the above example of learning manipulation places, the whole process needed 1,472,382 inference steps in Prolog and took 1.01s for a dataset of 140 observed manipulation actions. Since the model is cached afterwards, subsequent queries only need 17,565 inferences and 0.04s. The high performance is achieved by using direct computation instead of complex inferences whenever possible, by caching results, and by modeling in a way that supports the most common inferences.

B. Locating Objects based on their Function

When a robot is looking for objects, it usually needs them for a task it is about to perform. By combining the enviroment map with encyclopedic and common sense knowledge, the robot can query for objects by their functionality. The required knowledge about actions an object can be used for is partly already available from Cyc, partly imported from the OpenMind Indoor Common Sense (OMICS) database [13]. OMICS contains, amongst other things, detailed information about possible uses of objects that we imported into the knowledge base.

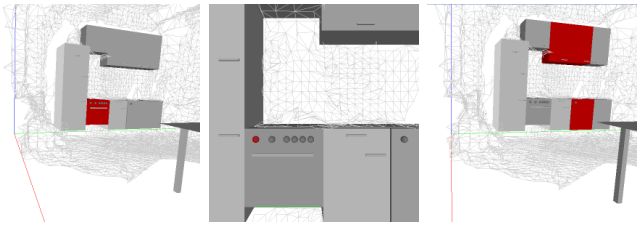

Fig. 5. Results of queries for objects that can be used for cooking food (left), for parts of the oven that cause a *Boiling* event (center) and for objects that contain drinking vessels (right), visualized by the graphical user interface of the knowledge processing system.

The left picture in Figure 5 is the result of a query for an object the robot can use for *CookingFood*:

```
owl_restriction_on('CookingFood', usesDevice, some(?T)),
owl_query(?S, type, ?T).
```

The encyclopedic knowledge base returns the concept *Oven* as binding for the variable *?T* and the semantic map locates an oven as displayed in the image. In order to operate the oven, the robot has to know which part to manipulate to cause a heating process. The query is shown below, its result is visualized in the center image in Figure 5.

```
owl_query(?OVEN,    properPhysicalPartTypes, ?KNOB),
owl_query(?OVEN,    type, 'Oven'),
owl_query(?KNOB,    causes-Underspecified, ?HEATING),
owl_query(?HEATING, postEvents, ?BOILING),
owl_query(?BOILING, type, 'Boiling').
```

The robot can also query the current state of objects, like cupboards that currently contain cups. The positions of cups are provided by RFID tag readers inside the cupboards.

```
owl_query(?O, in-ContGeneric, ?S),
owl_query(?O, type, ?T),
owl_query(?T, subClassOf, 'DrinkingVessel').
```

C. Reasoning about Actions

In Figure 1 we already introduced the hierarchical action model. The raw input sequence of human pose vectors is first segmented using Conditional Random Fields, which shall not be described further in this paper. The resulting motion segments are the basis for more complex actions, which are generated autonomously by matching action specifications against the track of observed motions.

Using the model, the system can perform inference on higher levels of abstraction, e.g. for selecting all actions with a certain purpose or all actions performed on the same kind of object. The results are all linked to the low-level datastructures describing the human poses while performing these actions. For instance, a query for all poses during a table setting episode is shown below and visualized in Figure 6 (left).

```
owl_query(?A, type, 'SetTable'),
postureForAction(?A, ?Posture)
```

By combining the queries with other information, like the objects involved, it is possible to select in a more fine-grained way, for instance the motion for taking a plate out of the cupboard (Figure 6 right).

```
owl_query(?A, type, 'TakingSomething'),
owl_query(?A, objectActedOn, ?O),
owl_query(?O, type, 'DinnerPlate'),
postureForAction(?A, ?Posture)
```

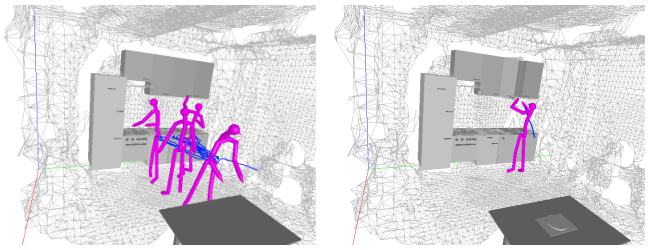

Fig. 6. Human pose sequences for setting a table (left) and taking a plate out of the cupboard (right).

D. Completing Underspecified Instructions

Instructions given by humans rarely contain enough information to be executed directly, but rather require a large amount of common-sense knowledge to be understood by a robot. Examples of missing pieces of information are the exact position and orientation of the knife in a command like "put the knife left of the plate" or the fact that "in front of the chair" in a table-setting context actually means "in front of the chair *and on the table*". Much of this information can be inferred by combining action models with the environment map. The correct orientation of objects can be learned depending on the object type and the position in the environment, for example on a table or inside a cutlery tray:

```
actionmodel(
    observables(objectClass, objX, objY, objZ)
    predictables(orientation))
```

The fact that items are to be placed on top of the table can be inferred using the concept of supporting entities: When the robot knows that each object has to rest on another entity, it can query the environment model and learn a relation between the region, the upper object and the most probable supporting entity:

```
actionmodel(
    observables(objectClass, objX, objY)
    predictables(supportingEntity))
```

Being able to understand and generate qualitative descriptions like "on the table" or "inside a cupboard" is crucial when interacting with humans. For resolving them to metric values, the system uses computable Jython properties that implement spatial heuristics. Relations like *on* or *inside* are calculated based on the positions and dimensions of the objects they relate. For instance, if the outer coordinates of the inner object are completely contained by the outer object, the *inside* relation holds.

V. Conclusions

In this paper, we present a practical knowledge processing system for mobile robots and sensor-equipped environments. It combines formal, encyclopedic knowledge with observations from several perception modules (a 3D environment map, human pose tracking, object recognition, and a rich sensor network) and directly works on the data structures produced by these perception modules.

We introduce the concept of computable classes and properties which serve for loading observations into the system and thus provide the link between the continuous sensor data and the symbolic concepts.

The action models described in this paper are learned based on abstract specifications and allow for discovering a class structure in observed data. They effectively extend the class taxonomy with learned, action-related concepts. We demonstrate this approach with learning specializations of *ManipulationPlace*.

In contrast to many toy-world knowledge representations, this system works on real observations of household tasks performed by humans or robots in a real environment.

Most of the requirements we introduced in the beginning are met by the system: The knowledge is organized in an action-centric way so that the robot can easily access the concepts it needs for its tasks. It is grounded in the perception and action system, and the knowledge grows and adapts as more observations are made. Since action models are learned on demand when needed, they automatically adapt to changes in the world. Inference tasks are performed very fast compared to the time usually required for manipulation.

VI. Acknowledgments

This work is supported in part within the DFG excellence initiative research cluster *Cognition for Technical Systems – CoTeSys*, see also `www.cotesys.org`.

References

[1] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira, "An introduction to the syntax and content of Cyc," *AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pp. 44–49, 2006.

[2] S. Harnad, "The symbol grounding problem." *Physica D*, vol. 42, pp. 335–346, 1990.

[3] N. Mavridis and D. Roy, "Grounded Situation Models for Robots: Where words and percepts meet," in *IROS*, 2006, pp. 4690–4697.

[4] A. Saffiotti *et al.*, "The PEIS-Ecology project: Vision and results," in *IROS*, 2008, pp. 2329–2335.

[5] I. H. Suh *et al.*, "Ontology-based Multi-Layered Robot Knowledge Framework (OMRKF) for Robot Intelligence," in *IROS*, 2007, pp. 429–436.

[6] M. Beetz *et al.*, "The Assistive Kitchen — A Demonstration Scenario for Cognitive Technical Systems," in *RO-MAN*, 2008.

[7] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.

[8] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3D Point Cloud Based Object Maps for Household Environments," *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 2008.

[9] U. Klank, M. Z. Zia, and M. Beetz, "3D Model Selection from an Internet Database for Robotic Vision," in *International Conference on Robotics and Automation (ICRA)*, 2009.

[10] R. B. Rusu, B. Gerkey, and M. Beetz, "Robots in the kitchen: Exploiting ubiquitous sensing and actuation," *Robotics and Autonomous Systems Journal (Special Issue on Network Robot Systems)*, 2008.

[11] J. Bandouch, F. Engstler, and M. Beetz, "Accurate human motion capture using an ergonomics-based anthropometric human model," in *Fifth International Conference on Articulated Motion and Deformable Objects (AMDO)*, 2008.

[12] A. Kirsch, "Integration of programming and learning in a control language for autonomous robots performing everyday activities," Ph.D. dissertation, Technische Universität München, 2008. [Online]. Available: http://mediatum2.ub.tum.de/node?id=625553

[13] R. Gupta and M. J. Kochenderfer, "Common sense data acquisition for indoor mobile robots," in *National Conference on Artificial Intelligence (AAAI-04*, 2004, pp. 605–610.