**Journal of Intelligent Service Robotics**
(preprint of DOI 10.1007/s11370-012-0118-y)

# Caesar – An Intelligent Domestic Service Robot

**Stefan Schiffer · Alexander Ferrein ·
Gerhard Lakemeyer**

Received: 23 April 2012 / Accepted: 6 September 2012

**Abstract** In this paper we present CAESAR, an intelligent domestic service robot. In domestic settings for service robots complex tasks have to be accomplished. Those tasks benefit from deliberation, from robust action execution and from flexible methods for human-robot interaction that account for qualitative notions used in natural language as well as human fallibility. Our robot CAESAR deploys AI techniques on several levels of its system architecture. On the low-level side, system modules for localization or navigation make, for instance, use of path planning methods, heuristic search, and Bayesian filters. For face recognition and human-machine interaction, random trees and well-known methods from natural language processing are deployed. For deliberation, we use the robot programming and plan language READYLOG, which was developed for the high-level control of agents and robots; it allows to combine programming the behaviour with using planning to find a course of action. READYLOG is a variant of the robot programming language Golog. We extended READYLOG to be able to cope with qualitative notions of space frequently used by humans such as "near" and "far". This facilitates human-robot interaction by bridging the gap between human natural language and the numerical values needed by the robot. Further, we use READYLOG to increase the flexible interpretation of human commands with decision-theoretic planning. We give an overview of the different methods deployed in CAESAR and show the applicability of a system equipped with these AI techniques in domestic service robotics.

**Keywords** Domestic Service Robotics · Situation Calculus · Golog · Decision-theoretic Planning · Qualitative Spatial Representation · Reasoning · Cognitive Robotics · Fuzzy Sets

Stefan Schiffer · Gerhard Lakemeyer
Knowledge Based Systems Group
RWTH Aachen University, Aachen, Germany
E-mail: {schiffer,gerhard}@cs.rwth-aachen.de

Alexander Ferrein
FH Aachen University of Applied Sciences
Aachen, Germany
E-mail: ferrein@fh-aachen.de

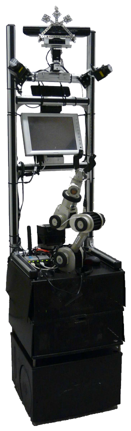
## 1 Introduction

To move to a retirement home should be the last step for an elderly person who cannot cope with her or his every-day tasks at home any longer. Living in your well-known and familiar home for as long as possible is an invaluable increase in living quality when you get older. Every-day tasks such as doing the chores or the groceries might become harder, however, being able to care for oneself is very important to keep independence and dignity. Being able to still participate in social life yields another big gain in living quality. Western societies face the problem of growing older and older meaning that more effort has to be put into the care of an ever growing older society. Besides caring for our elderly by ourselves, assistance systems for every-day tasks will become more and more important. Robot systems for unloading the dish-washer, being our pets (e.g. the pet robot Paro [39]) or autonomous cars (e.g. Stanley [42]) which will allow us to participate in social life and go to the cinema or visit the doctor's by ourselves might become a common thing in the future like a TV set is today.

For this to become reality, our robot needs to fulfil very complex tasks and requires sophisticated techniques, many of them from the field of Artificial Intelligence research. A robot living together with human beings helping them in their daily lives needs to be robust, fail-safe, understand the human and needs to exhibit intelligent behaviour. For the robot applications today this means that a lot of different capabilities need to be integrated into the robot system. As the robot is operated by a laymen and not by a robot expert, an intuitive human-machine interface is of utter importance. Of course, the robot needs basic abilities such as to localize itself in the domestic environment, it needs to navigate safely, and it needs to perform the given tasks in an intelligent way.

In order to bring forward the development of such domestic robots, there is a number of initiatives and research in domestic robotics going on. One among them is the RoboCup@Home competition, a competition inside the RoboCup initiative, which particularly focuses on domestic robot applications. A robot system in the competition has to fulfil a number of preset challenges such as seeking lost objects or being a robot butler recognizing people and seating them in the living room. In this paper, we present our intelligent domestic service robot Caesar which successfully participated in RoboCup@Home competitions. It features a solid base in low- and mid-level capabilities that have been developed and tailored to a competitive system in recent years. Various artificial intelligence techniques have been deployed at multiple levels. It has a logic-based high-level control that allows for flexible deliberation and decision making as well as robust behaviour and cognitive, user-friendly human-robot interaction.

The rest of the paper is organized as follows. In the next section we introduce the domestic service robot domain and present our domestic service robot Caesar. Then, we briefly review basic modules and software components for human-robot interaction. In Section 3, we present the high-level capabilities of Caesar deploying the logic-based high-level control language Readylog. After a review of related approaches, we conclude with a discussion and an outlook on future work in Section 5.


(a) Caesar, our Domestic Service Robot

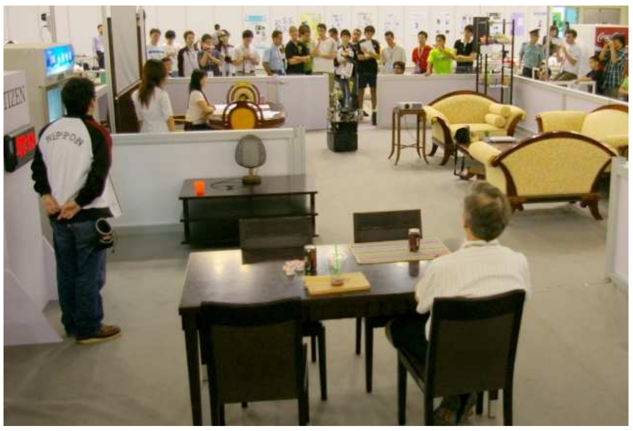

(b) The arena from the 2008 RoboCup@Home competition with Caesar performing the *WhoIsWho* test in it looking for people

**Fig. 1** Caesar and the arena of the 2008 RoboCup@Home competition

## 2 Caesar and the Domestic Service Robot Domain

In this section, we first introduce the domestic service robot domain before we outline Caesar's hardware and software system. We sketch the base components (navigation and localization) before we review modules for human-robot interaction (speech, face and gesture recognition).

### 2.1 The Domestic Service Robot Domain & RoboCup@Home

In this paper, we focus on intelligent domestic service robots, that is, autonomous robots that exhibit intelligent behaviour using artificial intelligence techniques. Our robot Caesar is shown in Fig. 1(a). It is designed and built to operate in human-populated environments, being helpful around the house, ultimately assisting elderly or disabled people with their daily activities.

Apart from the common requirements for any autonomous mobile robot, its application in domestic service robotics (DSR) scenarios with humans around places additional demands. Two very important issues are flexible and robust intelligent behaviour and human-robot interaction. When humans and a robot share the same habitat and are working together, the robot must exhibit some form of intelligent behaviour to be helpful for extended periods of time. At the same time it needs to be robust against various types of variations and errors in its environment, not only those caused by the humans. As an example for the former kind of variations consider that humans are messy (from a robot's perspective) and tend to leave things in different places. They move around items frequently so that the environment is not as predictable as, say, with industrial settings. For the latter kind of variations and errors, please recall that a robotic system for complex tasks is a complex system itself. Modules might crash and components might get stuck in certain situations. Robustness against those problems allows for enduring autonomy which is crucial when the robot needs to assist a person over extended periods of time. A domestic service robot has to meet the cognitive demands that arise in daily chores, where complex problems sometimes require deliberation. Strictly predefined behaviours are prone to errors and tend to fail since they cannot account for more or less subtle variations and the uncertainty inherent in real-world scenarios all the time. From a human-robot interaction perspective, robots need to be operable by laymen and they need to account for imprecision and fallibility of humans, in general, and of elderly people, in particular.

There are various efforts for research in domestic service robotics. One of them is the RoboCup@Home competition which particularly focuses on the technologies and techniques for domestic service robots. In annual competitions, researchers from all over the world gather to showcase their latest developments in a number of preset challenges, which become more and more complex from year to year. Since 2006, the RoboCup@home initiative [46, 47] fosters research in artificial intelligence, robotics, and related fields under the roof of RoboCup. It specifically targets the application of autonomous mobile robots as assistive technology in socially relevant tasks in home environments. It is designed to be both, a scientific competition and a benchmark for domestic service robots [44]. The general idea in the @Home competition is to set up a home-like scenario that is as realistic as possible and to let robots perform a set of tasks in that environment. An example of the scenario is depicted in Fig. 1(b), which shows a scene where the robot is driving around in the apartment with the goal to find and to recognize people. The tasks are oriented towards real-life applications of domestic service robotics.

The tests in DSR, in general, and in RoboCup@Home, in particular, require working solutions to specific (sub)tasks such as localization, navigation, face recognition and others. What is even more important, a successful system has to fully integrate those capabilities to a complete system that also has means of deliberation to purposefully work towards achieving a particular mission in a home-like scenario. Our team participated quite successfully in these competitions since they were established in RoboCup@Home. We were able to win the world championship in 2006 and 2007, and became second in 2008. In 2009 we just missed the finals. We also won the RoboCup German Open competition in the @Home league in 2007 and 2008.

## 2.2 Hardware Platform

The mobile robot CAESAR that we developed for (domestic) service robotics and that we use in the ROBOCUP@HOME competitions is based on the platform that was formerly used in the AllemaniACs RoboCup Team for the MIDDLE-SIZE league until 2006. It features several improvements dedicated to the specific requirements in service robotics.

CAESAR's base platform has a size of 40 cm × 40 cm × 60 cm (Fig. 1(a)). It is driven by a differential drive, the motors have a total power of 2.4 kW and are originally developed for electric wheel chairs. The power for the motors is supplied by two 12 V lead-gel accumulators with 15 Ah each. The battery power lasts for approximately one hour continuous motion at full charge. This power provides us with a top speed of 3 m/s and $1000^\circ$/s at a total weight of approximately 60 kg. Our main sensor for navigation and localization is a $360^\circ$ laser range finder (a Lase

ELD-L-A[1]) with a resolution of 1° at a frequency of 10 Hz. For communication a WLAN adapter capable of using IEEE 802.11a/b/g is installed.

Starting with robotic soccer, for which the basic platform was initially designed, we modified the hardware platform and extended the actuators and computing power to meet the requirements of the domestic environment. We report about beginnings and the transition from football to the domestic environments in [36]. One of the major modifications was the installation of an anthropomorphic *robotic arm* called Katana6M180[2] from Neuronics which we use for manipulation tasks since 2007. The Katana is equipped with six motors providing five degrees of freedom. The arm's weight is around 4 kg and it has a maximal payload of 500 g. The arm is mounted on top of the mobile robot platform described above. To provide the arm with the required power, we mounted two additional 12 V NiMH accumulators on the robot.

Together with the arm we added an aluminium rack to mount additional items such as stereo speakers to play sounds and to generate speech output as well as a ten inch touch screen to display information and to receive commands. The rack increases the total height of CAESAR to 180 cm. On the very top of the robot we recently installed a Microsoft *Kinect* sensor. It is used for visual servoing of the arm as well as for face, gesture and object recognition. The camera has a field of view of 57° horizontally and 43° vertically. It can be tilted in a range of ± 27°. The camera provides 32-bit colour images with a resolution of $640 \times 480$ and 16-bit depth information with a resolution of $320 \times 240$ both at 30 frames/s. The depth sensor ranges from 1.2 m to 3.5 m. The Kinect further also provides a 16-bit audio stream at 16 kHz.

To increase flexibility and to better direct the robot's gaze for a particular task, the camera is mounted on a self-assembled *pan-tilt unit (PTU)*. To further improve on our sensing capabilities we additionally installed two URG-04LX-UG01 *laser range finders* (LRF) from Hokuyo.[3] They provide distance data in a range of up to 5.6 m in a 240° scan window at a resolution of 0.36° with 10 scans per second. To meet the increased demands in computational power for vision, manipulation and deliberation we installed two Intel® Core™2 Duo computers running at 2 GHz with 2 GB RAM each.

## 2.3 Base Software Components

We now briefly review some of the base software components that are deployed on CAESAR. We mention the AI techniques that are used, but we do not go into detail on any of the particular modules and refer to the corresponding papers for a thorough description instead.

### *2.3.1 Software Framework*

We are using the robot control framework Fawkes [32] for many of our components. Fawkes is open source and made freely available.[4] Fawkes follows the

[1] `http://www.lase.de/produkte/2dscanner/eld_l_a/en.html`
[2] `http://www.neuronics.ch/cms_en/web/index.php?id=244`
[3] `http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html`
[4] `http://www.fawkesrobotics.org/`

component-based software design paradigm by featuring a clear component concept with well-defined communication interfaces. Each functional entity in Fawkes is a component, which can make use of several pre-defined aspects. Each component, implemented as a plugin that can be loaded at run-time, needs to inherit a communication aspect to communicate with other components. Plugins are realised as threads in a monolithic process. However, distributed design is possible by synchronizing the data between multiple instances via a network protocol with a minimal timing overhead. Via synchronization among the components we ensure that no superfluous work is performed. Fawkes comes with a number of useful plugins for tasks like timing, logging, data visualization, and software configuration. These make it particularly easy to create and to debug productive code, shortening the typical development cycle for robot software.

### *2.3.2 Navigation & Localization*

A mobile robot, especially when operating in human-populated environments, needs to avoid collisions with furniture or humans and it has to be safe in this regard. On CAESAR, we deployed a method for local navigation and collision avoidance that is safe and reactive [26]. It decomposes the task of local navigation into two steps. First, it plans a collision-free path from a start position to some goal position which is a target position relative to the robot. Then, it searches for feasible motion commands on that path to reach the target. CAESAR uses only a local map derived from the laser range finder's readings for its collision avoidance. Hence, it works in unknown environments. The dimension of the local map, a grid representation, corresponds to the area covered by the current reading from the LRF. To account for safety margins, we deploy a dynamic obstacle extension method. Detected obstacles are extended by safety margins computed from the current size and velocity of the robot. The faster the robot travels, the larger the obstacles that the robot senses will be represented in its local map decreasing the robot's clearance. In a first step, we use A*-search [24] to find an initial path to the target. The cost criterion used in the search is the distance to the target, that is, the length of the path. Having this path, in a second step we search for an approximation of the initial path taking into account the kinematic constraints of the robot, again using A*. We use the time to reach the target point as the cost function in this second search. Splitting up the problem into two independent search problems reduces the originally higher-dimensional problem to two problems with lower dimensionality as we decouple the path planning problem from the motion planning problem. The solution to the path planning problem then yields an initial solution to the subsequent motion planning problem. Based on the optimal path, motion commands to closely follow that path are sought for. Our method is very reactive and thus can easily account for dynamic changes. The collision avoidance module runs with a frequency of 20 Hz. It recomputes the path and the motion commands every cycle. Searching for a path of a length of up to 15 m takes less than 10 ms so that plenty of time remains for the subsequent steps of the algorithm. We deployed this approach for many years in robotics competitions both in robotic soccer as well as in domestic service robotics settings. It was a key to succeed in the domestic robot competition RoboCup@Home several years. For a thorough discussion of the method we refer to [26]. There, we also discuss some experimental results of the method. We compare the trajectories of our collision avoidance method with the well-known Dynamic Window Approach by Fox et al. [21] and show that our method computes smoother paths.

An intelligent robot has to know where it is located in its environment. Our self-localization uses a Monte Carlo Localization following [14]. It approximates the position estimation by a set of weighted samples. Each sample represents one hypothesis for the pose of the robot. Roughly, the Monte Carlo Localization algorithm now chooses the most likely hypothesis given the previous estimate, the actual sensor input, the current motor commands, and a map of the environment. To be able to localize robustly with the laser range finder, we modified the Monte Carlo approach. To allow for the integration of a whole sweep from the LRF, we use a heuristic perception model. With this we are able to localize with high accuracy both in the RoboCup environment as well as in home and larger scale office environments. The method is presented in detail in [40]. There we show in simulations that the position could be tracked very accurately even with a laser noise of up to 90 %.

On top of the metrical map used for localization, we additionally keep a topological map that stores information about the positions of and the connections between semantic places like a room or a door. To plan a path from one place to another we perform an A$^*$-search on the topological graph. This yields a list of waypoints along which the robot is to navigate to the target place. The local navigation between those waypoints is then performed by our collision avoidance method described above. Note that the collision avoidance method only works on a local map that is constructed from the laser data in each cycle. Therefore, collision avoidance works in completely unknown environments while the localization method requires a map of the environment.

For physical interaction with the world Caesar can not only move around, but it can also manipulate objects with its robotic arm. To do so, it perceives its surrounding with an RGB-D camera. From the depth information a local model of the scene is generated that is used in the motion planning for the arm. We use OpenRAVE [15] to plan collision free motion trajectories for the manipulator to make Caesar pickup objects and to place them somewhere else. For the inverse kinematics OpenRAVE pre-computes a database so that finding an inverse kinematics reduces to a look-up at run-time. The internal path-planning uses Bi-directional RRTs. For the manipulation tasks we encounter in our domestic scenarios, e.g. picking up a cup from a table with other objects on it also, the planning of a collision-free trajectory for the arm is done in below five seconds.

### 2.4 Human-Robot Interaction Components

Since the robot is to operate in a human environment, it needs capabilities to detect, recognize and interact with humans. We now briefly review the components on Caesar that enable human-robot interaction. The output of these modules is written to a central blackboard where other components and the deliberative robot high-level control can make use of the information currently available.

When working with humans, the robot has to be able to detect them and to tell them apart. Face detection and recognition is a feasible means to do so. Therefore, we employ an AdaBoost-based method for face detection [30, 43] which is readily

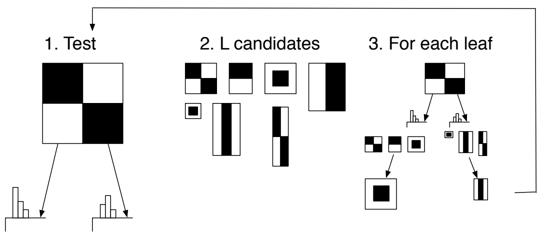


**Fig. 2** Schematic representation of the training method for face recognition.

available in OpenCV.[5] Further, we have developed an integrated approach [6] for face detection and recognition using random forests [8] where face recognition can also be used separately. The recognition framework is able to integrate new identities to its database on the fly. We use Haar features similar to those used in the boosted face detection cascade by Viola and Jones [43]. They allow for fast evaluation and they are known to be good features for face detection and recognition. We grow random trees (RT) for recognition in an iterative procedure. In each iteration, a set of $L$ so-called test candidates is randomly generated. A test is a particular Haar feature and a threshold value for that feature. At a node to grow further, the test candidates are applied to all the training samples, that is, to image patches labeled with the true class (i.e. the correct face identity). We measure the entropy gain when splitting with every test candidate (like in decision tree learning). The best candidate is chosen and the left and right branches are appended to the existing node; the procedure continues until the leaf nodes maintain training data of only a single class. The training process is schematically shown in Fig. 2. The training is very fast and the recognition performance of the resulting random forests is usually sufficient for our target scenarios. For example, it takes only about 7.5 ms to create a random forest consisting of ten random trees, each grown up to a 1000 nodes on a training collection of 464 face images and six identities to yield a recognition accuracy of over 85 %. For a more detailed description and further results we refer the interested reader to [6].

Spoken commands can be given to our robot Caesar conveniently using natural language. We use the Sphinx[6] speech recognition software system from Carnegie Mellon University. It is based on Hidden Markov Models (HMM). An overview of an early version of Sphinx is given in [25]. We have built a robust speech recognition system [16] on top of Sphinx by first segmenting closed utterances potentially directed to the robot. These utterances are decoded with two different decoders which run in parallel. One decoder uses a finite state grammar (FSG), the other one is a TriGram decoder. Applying both in parallel, we seek to eliminate each decoder's drawbacks retaining its benefits. We can look for similarities between the FSG's output and the $N$-best list of the TriGram decoder. This allows for highly accurate recognition in restricted domains such as RoboCup@Home. At the same time, false positives, which are immanent in the noisy environments one is confronted with at RoboCup competitions, can be filtered out reliably. A detailed evaluation of our method can be found in [16].

[5] `http://opencv.sourceforge.net/`

[6] `http://cmusphinx.sourceforge.net/`

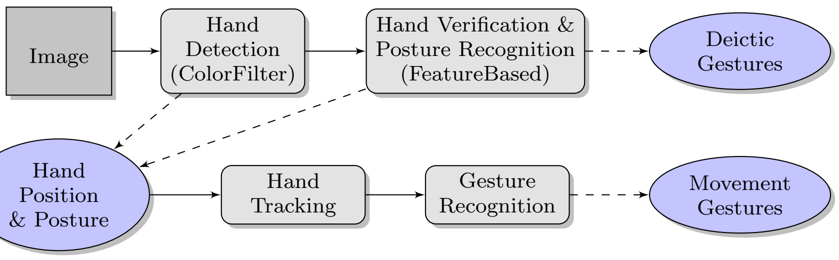


**Fig. 3** Architectural overview of our modular gesture recognition system

Speech recognition provides an intuitive means to control and interact with a robot. However, a huge part of meaning in communication is also transferred via non-verbal signals [17]. A very important mode of this non-verbal communication is using gestures. This is especially true in interaction with a domestic service robot, since controlling the robot often relates to entities in the world such as objects and places or directions. References to objects can conveniently be made by pointing gestures while other, dynamic gestures can be used to indicate directions or commands.

In Caesar's gesture recognition system, we implement a modular architecture [34] where the overall process is decomposed into sub-tasks orchestrated in a multi-step system. This enables a filter-and-refine like processing of the input where the single steps are as independent as possible to allow for an easy replacement of the particular method used for any of them. Further, the output of each step can already be used for specific purposes. That is, pointing gestures can already be inferred from the position (and possibly the posture) of a hand and a face, while recognition of dynamic gestures additionally requires hand tracking to extract a trajectory. What is more, the overall computational demands are kept low because the amount of information to be processed gets reduced in each step. While we use data from the Kinect sensor for hand detection, the gesture recognition is based on a modified version of the approach presented in [45]. The system performs reliably well in our target scenarios with an accuracy of up to 90 %. For a detailed account and evaluation results of the individual elements we refer to [34]. A graphical overview of the gesture recognition system architecture is depicted in Fig. 3.

## 3 Robot High-level Control and Human-Robot Interaction

So far, we sketched the base modules of our intelligent service robot. With all these components in place, we still need to have a way to specify high-level controllers for Caesar and to allow for deliberation where necessary, especially in domestic domains. For the specification of Caesar's high-level control we opt for a logic-based approach, namely the robot programming and plan language Readylog that we detail in the next section. After introducing the language, we show how it can be used both—for programming the behaviour of Caesar and also for planning to achieve deliberation. We exemplarily show how Caesar can deploy planning in DSR tasks beneficially. In Section 3.2, we discuss an extension of Readylog that allows for a (limited) form of self-maintenance. Subsequently in Section 3.3,

**Algorithm 1:** A Readylog program making use of the qualitative positional notions for the "*Fetch&Carry*" task. We omit some specification details to retain reasonable clarity.

```
proc fetch_and_carry($obj$)
  if $\exists x, y.\Phi(obj) \wedge pos(obj) = (x, y)$ then
    $(\pi\ x', y')[\Phi(obj) \wedge pos(obj) = (x', y')?;$
      pickup($obj$) ;
      approach($x', y'$) ;
      putdown($obj$)]
  else
    fail
  endif
endproc
proc approach($x, y$)
  if $\exists x_1, y_1, \text{is}(dist(x, y, x_1, y_1), \mathsf{close})$ then
    $(\pi\ x_1, y_1)[\text{is}(dist(x, y, x_1, y_1), \mathsf{close};\ \text{goto}(x_1, y_1)];$
  else
    fail
  endif
endproc
```

we show how we also use Readylog for human-robot interaction: to interpret commands issued to Caesar using natural language. Finally, we review a second extension of Readylog enabling us to represent and reason about qualitative spatial notions to bridge the gap between the robot and human users in Section 3.4.

## 3.1 High-Level Control with Readylog

Right off, we start the description of Caesar's high-level control with an illustrative example program from the RoboCup@Home competitions. Alg. 1 shows a program that can be used for the *Fetch&Carry* task in the RoboCup@Home competition. The task here is to assist the human instructor and seek, find and deliver some item that is located somewhere in the apartment. The program is written in Readylog, the control and planning language we deploy on Caesar [18, 20]. Readylog is a Golog dialect [29] and we particularly developed Readylog to extend Golog to be able to cope with real-time dynamic robot domains such as robotic soccer or the domestic service robot domain. Readylog borrows ideas from other Golog dialects [7,13,22,23,29] and provides a software architecture which allows for passive sensing, a mechanism to progress the internal database and allows for execution monitoring. This is particularly useful for uncertain domains when planning is used. It often happens that the policy that the robot has planned to achieve its mission goals is no longer executable due to unforeseeable changes in the world. With execution monitoring capabilities, the robot can at least find out that it can no longer accomplish its goals and restart the planning process. Alg. 1 should only give a flavour of Readylog programs in the domestic domain. In the following, we will briefly outline the program and, for now, leave out some details about the qualitative positions that are used in the program. We will come back to these in detail later in Section 3.4.

The program takes the name of the object of interest as argument. At first sight, the program looks like a hard-coded imperative program for the task. On closer inspection, it reveals some statements that are not so common. In line 3 of Alg. 1, we use a so-called **pick**-statement (the $\pi$) to retrieve the coordinates from the object of interest from our items database $\Phi(obj)$. $\Phi(obj)$ is a formula which encodes all objects in our domain that we are interested in, grounding their symbols. Each object position is then encoded via the fluent *pos*, which is a relation that can change over time as we define below. Thus, the position of each object can change over time. We simplified the program slightly and left out some technical details about the qualitative representational frames for readability reasons. All technical details can be found in [35]. Once we identified the object in question, we pick up the object at its location and approach the target position. There, in lines 12–13 we make use of our qualitative world model and try to get "close" to the target location. We give some more details about this in Section 3.4. This program gives a first idea of how Readylog programs look like. Some further examples of Readylog programs for domestic domains are shown below.

The semantics of the different language constructs are formally defined in the situation calculus, a second order language with equality which allows for reasoning about actions and their effects [31, 33]. We will not go into the details of the semantic definition of the language features here and refer to [18, 20] for a concise overview. In the following, we will just give an informal overview of the different constructs featured by Readylog. In the situation calculus, the world evolves from an initial situation due to primitive actions. Possible world histories are represented by sequences of actions. The situation calculus distinguishes three different sorts: *actions*, *situations*, and domain *objects*. A special binary function symbol $do : action \times situation \rightarrow situation$ exists, with $do(a, s)$ denoting the situation which arises after performing action $a$ in the situation $s$. The constant $S_0$ denotes the initial situation, i.e. the situation where no actions have occurred yet. The state the world is in, is characterized by functions and relations with a situation as their last argument. They are called *functional* and *relational fluents*, respectively. The third sort of the situation calculus is the sort *action*. Actions are characterized by unique names. For each action, one has to specify a *precondition axiom* stating under which conditions it is possible to perform the respective action and an *effect axiom* formulating how the action changes the world in terms of the specified fluents. All these axioms together with some foundational axioms form a so-called *basic action theory (BAT)*. For details we refer to [33].

### *3.1.1 Language Features*

Besides test actions, conditionals, loops, recursive procedures known from imperative programming languages, Readylog offers some non-standard constructs which are particularly useful for solving complex tasks in real-world applications. In the following, we give some examples of such non-standard constructs.

As an extension to well-known flow-of-control statements, Readylog offers a number of statements for executing actions in parallel or interrupt-triggered and guarded execution. These concepts together allow to encode complex control programs.

- $\sigma_1 \,||\, \sigma_2$: *Prioritized execution*
  $\sigma_1$ and $\sigma_2$ are executed in parallel, with $\sigma_1$ being executed first w.r.t. execution time. The whole statement terminates if either $\sigma_1$ or $\sigma_2$ terminates.
- **withPol**$(\sigma_1, \sigma_2)$ : *Prioritized execution of* $\sigma_2$
  It is a shortcut for $(\sigma_1; false?)||\sigma_2$. The first branch can never terminate since the last test action $false?$ will never terminate. Hence, this statement waits until $\sigma_2$ terminated as well.
- **withCtrl** $\varphi$ **do** $\sigma$ **endwithCtrl**: *Guarded execution*
  The program $\sigma$ is executed as long as the guard condition $\varphi$ holds true.
- **waitFor**$(\tau)$: *Event-interrupt*
  Somewhat adjacent to the previous statement, the event-interrupt pauses execution of program execution until a certain condition $\tau$ encoding some properties of the world becomes true.

Another interesting feature of Readylog are probabilistic projections. With a statement **prob**$(p, \sigma_1, \sigma_2)$ it is possible to assign a probability $p$ to program $\sigma_1$ denoting the likelihood that $\sigma_1$ will succeed. With probability $(1 - p)$, program $\sigma_2$ will succeed. Note that we do not assign probabilities to single actions, but to complex programs. This is quite powerful as we can assign probabilities to rather complex behaviours. Now, with the statement **pproj**$(\varphi, \sigma)$ we can check if the condition $\varphi$, which could be a complex formula over fluents, holds after simulating the program $\sigma$. With this feature which was first proposed in [23], we can encode goal-directed behaviour under uncertainty.

Finally, we want to sketch the **solve** statement which allows for decision-theoretic planning in the Readylog framework. To this end, the **solve**$(h, f, \sigma)$ statement initiates decision-theoretic optimization over $\sigma$ up to a fixed horizon $h$. Readylog implements a version of forward-search value iteration as proposed in [7]. Fig. 4 shows an example decision tree. The result of this statement is an optimal policy that can then be executed. This mechanism can also be used to find optimal arguments of an option. The **pickBest**$(h, \{x_1, \ldots, x_n\}, \sigma)$ statement optimizes the program $\sigma$ for a finite set of possible arguments $x_1, \ldots, x_n$. Basically, for each argument a branch in the forward-search tree is expanded (similar as presented in Fig. 4) and the optimal policy over all the $x_i$ is sought for. To illustrate this, we show the optimal policy for a simple navigation task in a maze and its Readylog formulation in Fig. 5. The procedure *navigate* invokes a **solve** statement, which optimizes over the basic actions *go_left*, *go_right*, *go_up*, *go_down* as long as the robot's current position is unequal to the goal position. The "|" is used to denote the action alternatives in the code given in Fig. 5(b). The resulting policy is a conditional Readylog program, which then can be executed (see [19] for the details how the policy is executed).

In the following, we give two examples of how Readylog can be applied in the DSR domain. In particular, we will show how some of the language features described above can be used in a beneficial way to encode the robot's high-level behaviour in a programming and a planning example. Note that many of the underlying programming concepts are, of course, also available in other programming languages. Integrating them into Readylog, however, has several advantages. One is that the domain description follows a BAT. Actions with their preconditions and effects allow for a very natural way to encode the robot's behaviour. A second ad-

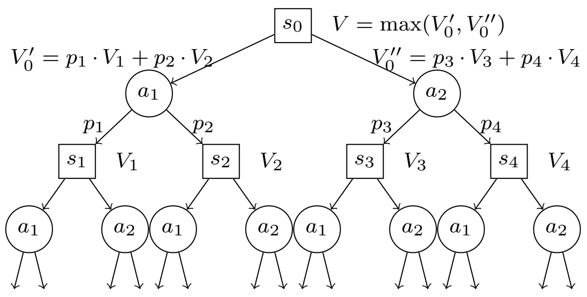
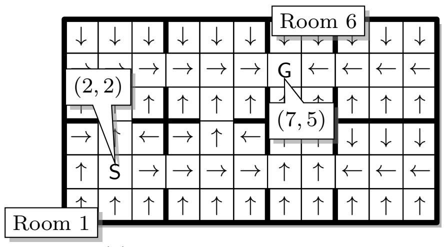


**Fig. 4** Decision tree search in READYLOG representing a non-deterministic choice of actions. In each situation $s_i$ the agent may choose between $a_1$ or $a_2$. The resulting optimal policy is represented by the branch with maximal value $V$.



(a) The Maze66 domain.

```
proc navigate
  solve(h, reward, while loc ≠ goal do
      (go_right | go_left | go_up | go_down)
  endwhile)
endproc
```

(b) The READYLOG code for navigation.

**Fig. 5** Navigating task solved with READYLOG

vantage is that the language has a formal semantics. This, in particular, allows for verifying READYLOG programs formally.

### *3.1.2 Behaviour Specification with Programming*

If the task that CAESAR should perform is straightforward and the course of actions can be specified in advance, the easiest way to do so is to program the behaviour step by step. A situation where this is applicable is the "Robot Inspection and Poster Session" test (RIPS) from the RoboCup@Home competitions. In this test, the robot is supposed to register itself and its team by entering the arena and by handing in a registration form with the Technical Committee (TC). The robot can then say a few words about itself and it can introduce the team as well. While the robot leaves after it introduced itself, someone from the TC tests whether the emergency kill switch is is working order. Alg. 2 shows the READYLOG programs used in the RIPS test. Since this test does not require much of deliberation, we only need the deterministic programming features provided by READYLOG. We can simply specify the robot's behaviour in the program by issuing the skill calls that correspond to the actions we want the robot to execute.

### *3.1.3 Using Planning – Order Cups Demo*

Sometimes, pure programming is not enough to specify the robot's behaviour, because the course of actions is not determined from the start. To demonstrate

**Algorithm 2:** READYLOG program for the "Robot Inspection and Poster Session" (RIPS) test.

```
proc rips
    grasp("Registration Sheet"), park_arm;
    ?(on = epf_start_button_status), %% wait for start
    goto_place("Registration Desk");
    introduce_yourself_and_the_team,
    extend_arm, give("Registration Sheet","TC");
    say("I'm going to leave through Door B now.");
    say("If you want, you can now test my emergency stop button.");
    goto_place("Door B"),goto_place("Door C");
    goto_place("Out C")
endproc
```

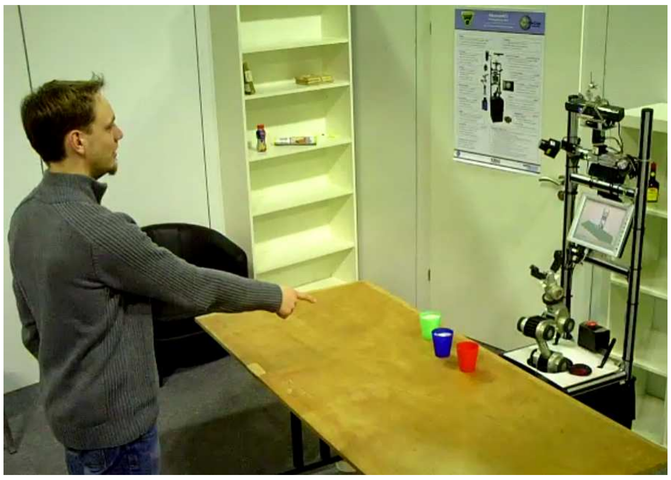


**Fig. 6** A user instructing the robot to sort cups with pointing gestures

how the reasoning capabilities provided by READYLOG can be put to good use, we discuss a special helping task that CAESAR is able to perform: the *Order Cups Demo (OCD)*. The robot is supposed to help setting the table. There are three differently coloured cups (red, green, and blue) on the table. They have to be put in a specific order to complete the re-arranging. A human user is instructing the robot on the desired order by pointing to positions on the table and by simultaneously specifying which cup should be placed at that very position. Once CAESAR has collected all the necessary information, it uses decision-theoretic planning to compute the optimal course of re-arranging the cups to reach the desired goal situation, i.e. CAESAR computes the re-ordering with a minimum number of movements. Alg. 3 shows the READYLOG programs used in the Order Cups Demo. Fig. 6 shows a user specifying the desired order of the cups by pointing.

The top-level program *order_cups_demo* is straight-forward: After perceiving the initial order of the cups on the table, CAESAR engages in interaction with the human user to inquire about the desired goal configuration of the cups on the table.

**Algorithm 3:** Readylog programs for the Order Cups Demonstration

```
proc $order\_cups\_demo$,
  %% perceive initial order
  $get\_Initial\_Order(I1, I2, I3, Init)$;
  %% inquire about goal order
  $get\_Goal\_Order(P1, P2, P3, Goal)$;
  %% set initial cup positions
  $set(cup\_pose(I1, I2, I3), (p1, p2, p3))$;
  %% start planner
  $sort\_cups(P1, P2, P3, 4)$;
  %% say that we're done
  $say$("That is it. I am done.")
endproc

proc $sort\_cups(P_1, P_2, P_3, H)$,
  solve$(H, reward\_cup(P_1, P_2, P_3)$,
    while$(\neg(p1 = pos(P_1) \wedge$
      $p_2 = pos(P_2) \wedge$
      $p_3 = pos(P_3)))$ do
      pickBest$(cup, \{red, green, blue\}$,
      pickBest$(to, \{p_1, p_2, p_3, p_4\}$,
        $move\_cup(cup, pos(cup), to)))$
      endwhile
  ) %%endsolve
  endproc
```

We do not discuss the details of the human-robot interaction procedure here but we only mention that it involves calling sub-routines that make use of the information provided by the modules described in Section 2.4. With this information it calls the procedure *sort_cups*. This procedure now uses decision-theoretic planning to sort the cups in an optimal way. It does so by using **pickBest** to select the next cup and a position to move the cup to in every step until all cups are at their desired goal positions. The optimization theory is mainly encoded in the reward function. In the case of the OCD task, the function gives a higher reward to situations with fewer actions performed by the robot. This way, Caesar computes and executes a policy to perform the re-ordering with a minimal number of movements.

### *3.1.4 Execution Monitoring*

One problem of applying Readylog policies to the real world is that all decisions are based on models of the basic actions theory, consisting of all the axioms that encode the action precondition, effects, etc. If a decision was taken at the time of planning and the condition no longer holds when the robot wants to execute the policy, it becomes invalid. For instance, consider a robot that plans to grasp a cup, and at the time of executing its policy, the cup was moved away by a human rendering the policy invalid. Such problems are inherent to off-line planning. However, we need to describe a mechanism where we at least can find out, when a planned policy becomes invalid.

We now sketch our solution, without going into any technical detail. These can be found in [20]. The execution monitoring makes use of discrepancies between the model assumptions made at planning time and the real-world situation that is given at execution time. So, in order to detect when there are discrepancies between certain model assumptions and reality, we store the assumptions directly in the policy. For example, if the program at planning time contained a statement such as ... **if** $\neg$*doorIsOpen* **then** *tryOpenDoor* | *callAssistance* **else** ..., then the crucial assumption for the executability of the policy that will be created from this program lies in the *doorIsOpen* predicate. To keep track of this crucial assumption, we introduce a marker $\mathfrak{M}(\varphi, v)$ in the policy that stores the truth value $v$ of the formula $\varphi$ at planning time. The resulting policy snippet might then be ... $\mathfrak{M}(\neg doorIsOpen, true)$; *tryOpenDoor*; *senseEffect*(*tryOpenDoor*); ... At run-time, the marker is re-evaluated based on the current state of the world.

If the stored truth value differs from the re-evaluated one, then the assumptions made at planning time do no longer hold and the policy is invalid. This way, the robot knows immediately, whether it can continue to execute the policy or if it should rather start re-planning. For the red cup in the *Order Cups Demo* this would mean that during off-line planning, a marker would have been introduced into the policy with the assumed position of the red cup. Consider that the cup was now removed just as the policy was going to be executed. First, the marker about the position of the red cup is re-evaluated. Now as the red cup was removed, a mismatch in the marker values is detected and re-planning is initiated.

While these markers allow for detecting when a policy becomes invalid, for robust execution control the robot also needs to be able to do introspection and to reason about its own capabilities. In the next section, we show our approach to increase robustness of program execution by introducing methods for self-maintenance.

## 3.2 Self-Maintenance for Increased Robustness

Monitoring the execution of actions to cope with changes in the environment is not sufficient to ensure robust robot behaviour. In order to make a robot execute a given task plan more robustly we further need to take care of the internal state of the robot. That is, we want to enable it to take care of its self-maintenance requirements during online execution of a plan. This requires the robot to know about the (internal) states of its components, and about constraints on the internal states that restrict execution of actions.

We developed a *constraint-based self-maintenance framework* [38], which enables an agent to monitor its self-maintenance requirements during program execution. We implement this as a transformation process on the plans formulated in READYLOG to be performed, based on explicit qualitative temporal constraints. Afterwards, a 'guarded' execution of the transformed program yields more robust behaviour. Whenever the framework determines unsatisfied requirements, appropriate recovery measures are performed online. This behaviour increases agent autonomy and robustness. Fig. 7 depicts how we integrate the components of our self-maintenance framework into the existing agent controller.

The program transformation uses explicitly formulated constraints that express dependencies between task actions and the robot itself. These are only important at run-time and we cannot and do not want to consider them at planning time already. The constraints that we consider are usually temporal, such as "the collision avoidance module must be running while the robot moves". To formulate these temporal constraints between actions, we obviously require a notion of temporal relations between actions (or more generally, between states). Since we are interested in constraints that should be easy to formulate for the designer, we prefer a qualitative description of these relations. We consider this sufficient for most cases we intend to handle and spare computing explicit timing values. We therefore chose Allen's Interval Algebra [2] as our basis.

We intervene between decision-theoretic planning, whose output is an executable program, and its online execution. Before any action of the program is performed in the real world, a self-maintenance interpreter checks whether there

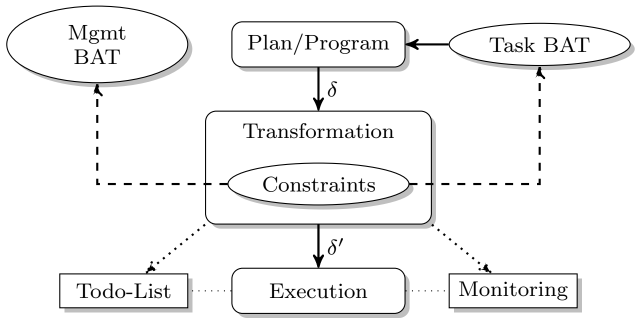


**Fig. 7** Architecture of our self-maintenance approach. The program $\delta$, which only uses the *Basic Action Theory* (BAT) for the task, is passed to our transformation process. This process uses constraints, which link elements from the Task BAT to elements from the self-maintenance domain. The latter are specified in the Mgmt BAT. The transformation yields a modified program $\delta'$ which is passed on for execution together with instructions for monitoring and commitments to future action. These commitments are maintained in a todo-list to ensure that they will eventually be satisfied.

are unsatisfied constraints for this action. If such constraints are found, the execution of this action and hence the execution of the program is delayed. We formulate a constraint satisfaction problem (CSP) for the set of unsatisfied constraints for the next action to be executed. We use the solution to this CSP to augment the program with maintenance and recovery measures such as starting modules which are currently not running but are supposed to. The augmented program is only then passed on for execution. Depending on the constraint(s) the transformation also includes monitoring markers, e.g., making sure the *locomotion_module is off* throughout the execution of *manipulating* something. It possibly also requires a list of commitments to actions in the future. [38] gives a more thorough presentation of our method.

## 3.3 Flexible Command Interpretation

The speech recognition system [16] for instructing our mobile robot mentioned in Section 2.4 makes use of a simple, restricted grammar where pre-defined utterances are directly mapped to system calls. This does not take into account fallibility of human users and only allows for binary processing; either a command is part of the grammar and hence understood correctly, or it is not part of the grammar and gets rejected. To increase the flexibility of our human-robot interaction, we model language processing as an interpretation process where an utterance issued towards the robot needs to be mapped to its capabilities on the fly. We do so by casting the processing as a (decision-theoretic) planning problem on interpretatory actions [37]. More precisely, we formulate the interpretation process as a basic action theory in Readylog.

We first analyse the given utterance syntactically by using a basic yet generic grammar. Then, we cast the interpretation as a planning problem where the single actions available to the planner are to interpret syntactical elements of the utterance. If ambiguities are detected in the course of the interpretation process, the system uses decision theory to weigh up different alternatives. The system is

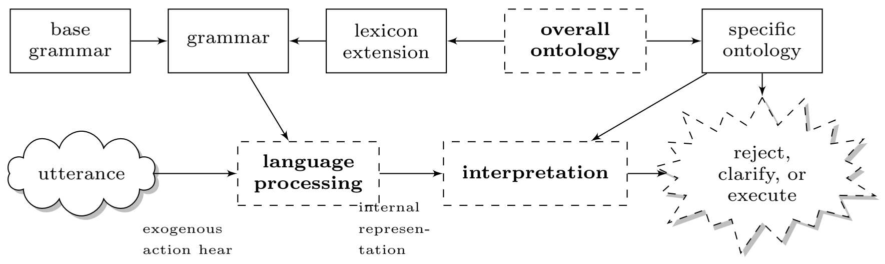


**Fig. 8** Overview of our language interpretation system

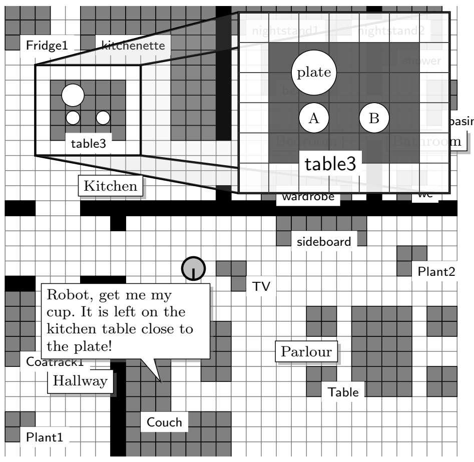
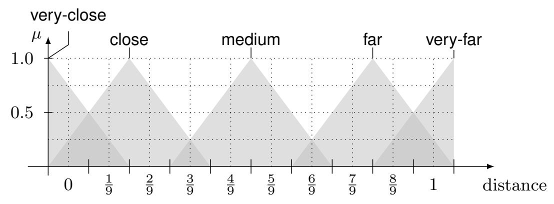


**Fig. 9** The domestic robot domain

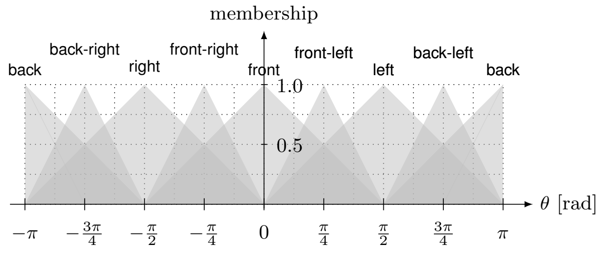
also able to initiate clarification to resolve ambiguities and to handle errors as to arrive at a successful command interpretation eventually. That is, the system inquires with the user about things like objects missing in the utterance and impossible combinations of actions and objects. For example, the utterance “Fetch the kitchen!” is not a valid request because rooms cannot be transported. Since our current high-level control already knows about the robot’s capabilities (the actions and the parameters that these actions need), we tightly connect the interpretation with it. We do so by interconnecting the basic action theory of the robot with the basic action theory of the interpretation process. An overview of the system architecture is given in Fig. 8. We tested our system with utterances of different complexity and also with different lexicon sizes. In realistic settings the robot is able to successfully interpret an executable request or to reject invalid requests with response times of below ten seconds. A more extensive discussion of our experimental evaluation is given in [37].

## 3.4 Qualitative World Modelling

In order to communicate with or instruct Caesar in a natural way, we use spoken language. Besides being able to process the language input at the speech recognition level (described in Section 2.4), we need to determine which ability of the



(a) Unit membership function for qualitative distance.



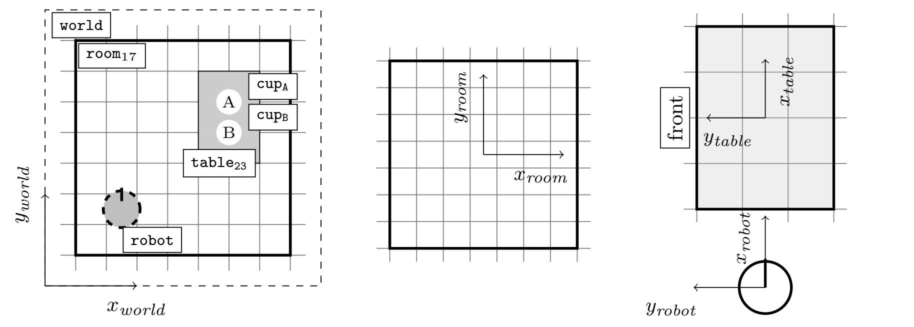
(b) Membership function for qualitative orientation at level of granularity 3.

**Fig. 10** Membership functions for distance and orientation in our domestic robot domain.

robot is being addressed by the user's request (as discussed in Section 3.3). Moreover, we also need to enable our robot to understand the objects referred to in our instructions on a semantic level. Our goal is to analyse and make the robot understand instructions such as "*get me the left cup on table2*" or "*bring me a coke to the living room*". Fig. 9 shows a typical domestic example. A closer look at such instructions shows that mainly qualitative positional information is used by the human instructor, i.e. qualitative distances and orientations. That means, we need to correctly interpret concepts such as "left/right" "near/far". Further, to be able to put the information given by the human instructor into the right context, we need a concept of a frame of reference possibly including a point of view. For example, "far" in the context of the living room refers to a larger distance than "far" in the bath room. To cope with these types of contexts and different points of view, we introduce the concept of *frames* and formalize them in the situation calculus.

First, we fix qualitative distance and orientation relations for our domestic environment. We define distance relation with the categories very-close, close, medium-far, far, and very-far. We define these qualitative concepts in terms of fuzzy fluents with possibly overlapping category boundaries as shown in Fig. 10 (see also [35] for the formal account of qualitative fluents in the situation calculus). Of course, in different frames, these categories have different scales. As the parlour is larger than the bath room, we have to scale these categories according to their respective frame. Our concept of *frame*, which we introduce shortly, is doing exactly that. For the qualitative orientation, we select a level of granularity of 3, meaning that we distinguish $2^3 = 8$ different qualitative orientations (Fig 10(b)).

In order to relate the orientation to its context, each frame needs to have a coordinate system with a maximal distance and a distinguished front side. Fig. 11 shows the basic idea. The most general possible frame is the world frame. Our



(a) World or global frame (b) Room frame (c) Table and robot frame

**Fig. 11** Examples of different frame in our domain.

domestic world consists of different rooms, each room has a world coordinate and an orientation in the world (Fig. 11(a)). In each room there is furniture and other objects and their coordinates are given relative to the room coordinate system (Fig. 11(b)), similarly, the table or the robot position (Fig.11(c)) is relative to the room coordinate. Object positions located on the table such as our example coffee cup would be given relative to the table coordinate system. In addition, we define a fixed front side of objects. In [35], we define a general scheme of how frames can be formally defined in the situation calculus and how the different object frames can be related to each other in order to come up with distinguished coordinates for the objects in our world.

An example of qualitative fluents was given in the *Fetch&Carry* example in Alg. 1. There, we made use of a qualitative distance fluent in the *approach* procedure. In lines 12 and 13 of Alg. 1 we check whether the distance between the target object and the robot is falling into the category "close" in order to slowly approach the front side of the respective object.

With these concepts we are able to interpret word phrases like "the left cup on the table close to the plate" in the correct way and are able to compute the right coordinates for the robot to really approach and grab the left cup which is located on the kitchen table. While we merely sketched the idea of the frame concept here, for a concise formal definition we refer to [35].

## 4 Related Work

There are various ways to achieve intelligent behaviour for autonomous mobile service robots. For a lot of specific sub-tasks, statistical methods are used. However, for the high-level control sometimes plan-based approaches come into play. We now briefly review some of the existing approaches in domestic service robotics related or alternative to the methods presented in this paper.

Some existing work uses formal languages to represent plans and control actions [3] and to integrate the high-level control with human-robot interaction components [4]. Well-founded formal basics allow for open-ended application of personal robots in human environments such as for every-day manipulation tasks [5]. The robot can cope with incomplete task specifications by filling the gaps and detailing abstract parts of the task on its own. To be able to cope with execution failures Lemai et al. [28] propose a framework for interleaving temporal planning and plan execution for robots. They are able to perform online plan repair and execution control based on temporal constraints. Their motivation is that "taking into account run-time failures and timeouts" requires online plan recovery.

Alami et al. [1] report on how they include the presence of humans in their robot control. Their robot is able to account for humans in the environment in its decision making in order to, for example, not interfere with a human watching TV by driving in the line of sight. The approach reported on in [9] uses classical AI planning techniques to determine a course of action to execute human user commands with taking into account the current situation. It can also revise its plan depending on updated perceptual input.

Stückler and Behnke lay out their development of a domestic service robot platform in [41]. They present their robot along three main issues: navigation, manipulation and communication. They focus on their robot's communication skills and do not go into too much detail on their high-level control mechanisms. In [10], Breuer et al. present their service robot for domestic domains and they detail the components of their system. It also features reasoning capabilities using an ontology and HTN planning. Chen et al. developed a service robot that uses a broad spectrum of AI techniques. To integrate natural language processing with action planning they use answer set programming [12]. To improve on their robot's abilities they also use techniques such as commonsense reasoning [11] and non-monotonic reasoning [27].

The approaches mentioned above present a selection of feasible solutions to particular challenges and some have favourable properties in their integration efforts. Nevertheless, we consider our system a coherent logic-based approach to the high-level control of an intelligent domestic service robot that has proven to be capable and successful in human-robot interactive environments.

## 5 Conclusion

In the introduction we drew a picture where domestic service robots will be part of our daily-life in a not too distant future. Besides a number of social and ethical implications which were not at all discussed in this paper, it seems that such personal assistance systems can bring a major gain in living an independent life in the older ages. Not being able to do the chores or groceries by oneself should not inevitably mean that people have to give up their self-determined life in their familiar home environment. Assistant robots could fill the care gap that might occur in the future in western societies. Robots that live together with humans need to understand the orders from their human instructors in the way humans communicate and they need to solve complex tasks in an intelligent way. Under the roof of the RoboCup Federation, there is the RoboCup@Home competitions which focuses on the development of domestic service robots. These robots are to fulfil complex tasks that occur in a domestic setting during the competition.

In this paper, we presented the autonomous robot Caesar which is our approach to the challenging domestic environment and we discussed the AI techniques that we use. We sketched the basic software system comprising of navigation and localization components. Of particular importance is human-machine interaction in such a setting. Therefore, we outlined the possibilities to robustly communicate and instruct the robot by gestures or speech. On the intelligent control side, we showed several examples of our high-level control component Readylog which features—besides imperative programming constructs—decision-theoretic planning and an account to modelling the world in a qualitative way. This is of importance as humans will interact with the robot by using qualitative instructions where distances and orientations will be given qualitatively such as "near/far" or "left/right". Moreover, humans tend to understand very easily the implicit scaling where "far" in the context of travelling from Europe to North America means something different than "far away" at the other side of the living room. We proposed the concept of qualitative frames which include scaling factors to account for these different scales. We showed several examples how Readylog could be used beneficially in the domestic setting. The robot system Caesar presented in this paper successfully participated in national and international RoboCup@Home competitions. Being able to rank first in both competitions twice also gives some indication about the benefits and usefulness of the system to the target audience.

For future work, we plan to integrate two more features to our robot. For one, we want to enable the robot to acquire new abilities in interactive communication with humans user. For another, we want to facilitate personalization of the robot to different users, e.g. by memorizing their individual characteristics of spatial concepts. For example, "high" for a small person might start earlier than for a tall person and "far" might be different for a person which has problems walking than for someone with good mobility.

**Acknowledgements** This work would not have been possible without the continuous efforts of the members of the AllemaniACs RoboCup Team, namely Tobias Baumgartner, Daniel Beck, Vaishak Belle, Masrur Doostdar, Niklas Hoppe, Bahram Maleki-Fard, Tim Niemüller, Christoph Schwering, Andreas Wortmann and others. We gratefully acknowledge partly funding for this research by the German National Science Foundation (DFG) under grants La 747/9-1, 2 and 5 and by the NRW Ministry of Education and Research (MSWF). We also thank the anonymous reviewers for their helpful comments on earlier versions of this paper.

## References

1. Alami, R., Clodic, A., Montreuil, V., Sisbot, E.A., Chatila, R.: Task planning for human-robot interaction. In: sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence, pp. 81–85. ACM, New York, NY, USA (2005)
2. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
3. Beetz, M.: Structured reactive controllers. Journal of Autonomous Agents and Multi-Agent Systems **2**(4), 25–55 (2001)
4. Beetz, M., Arbuckle, T., Belker, T., Cremers, A.B., Schulz, D.: Integrated plan-based control of autonomous robots in human environments. IEEE Intelligent Systems **16**(5), 56–65 (2001)
5. Beetz, M., Jain, D., Mösenlechner, L., Tenorth, M.: Towards performing everyday manipulation activities. Robotics and Autonomous Systems **58**(9), 1085–1095 (2010)
6. Belle, V., Deselaers, T., Schiffer, S.: Randomized trees for real-time one-step face detection and recognition. In: Proceedings of the 19th International Conference on Pattern Recognition (ICPR'08), pp. 1–4. IEEE Computer Society (2008)
7. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-theoretic, high-level agent programming in the situation calculus. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00), pp. 355–362. AAAI Press (2000)

8. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)
9. Brenner, M.: Situation-aware interpretation, planning and execution of user commands by autonomous robots. In: The 16th IEEE International Symposium on Robot and Human interactive Communication (RO-MAN 2007), pp. 540–545 (2007)
10. Breuer, T., Giorgana Macedo, G., Hartanto, R., Hochgeschwender, N., Holz, D., Hegger, F., Jin, Z., Müller, C., Paulus, J., Reckhaus, M., lvarez Ruiz, J., Plöger, P., Kraetzschmar, G.: Johnny: An autonomous service robot for domestic environments. Journal of Intelligent & Robotic Systems **66**, 245–272 (2012)
11. Chen, X., Ji, J., Jiang, J., Jin, G., Wang, F., Xie, J.: Developing high-level cognitive functions for service robots. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1, AAMAS '10, pp. 989–996. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2010)
12. Chen, X., Jiang, J., Ji, J., Jin, G., Wang, F.: Integrating nlp with reasoning about actions for autonomous agents communicating with humans. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02, WI-IAT '09, pp. 137–140. IEEE Computer Society, Washington, DC, USA (2009)
13. De Giacomo, G., Lésperance, Y., Levesque, H.: ConGolog, A concurrent programming language based on situation calculus. Artificial Intelligence **121**(1–2), 109–169 (2000)
14. Dellaert, F., Fox, D., Burgard, W., Thrun, S.: Monte Carlo localization for mobile robots. In: Proceedings of the International Conference on Robotics and Automation (ICRA) (1999)
15. Diankov, R., Kuffner, J.: Openrave: A planning architecture for autonomous robotics. Tech. Rep. CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA (2008)
16. Doostdar, M., Schiffer, S., Lakemeyer, G.: Robust speech recognition for service robotics applications. In: Proceedings of the International RoboCup Symposium 2008 (RoboCup 2008), *LNCS*, vol. 5399, pp. 1–12. Springer (2008). Best Student Paper Award
17. Engleberg, I.N., Wynn, D.R.: Working in Groups: Communication Principles and Strategies, 4 edn. Allyn & Bacon (2006)
18. Ferrein, A.: Robot controllers for highly dynamic environments with real-time constraints. Künstliche Intelligenz **24**(2), 175–178 (2010)
19. Ferrein, A., Fritz, C., Lakemeyer, G.: On-line decision-theoretic Golog for unpredictable domains. In: S. Biundo, T.W. Frühwirth, G. Palm (eds.) KI 2004: Advances in Artificial Intelligence, Proceedings of the Twenty-Seventh Annual German Conference on Artificial Intelligence, (KI-04), *Lecture Notes in Computer Science*, vol. 3238, pp. 322–336. Springer Verlag (2004)
20. Ferrein, A., Lakemeyer, G.: Logic-based robot control in highly dynamic domains. Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics **56**(11), 980–991 (2008)
21. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine **4**(1), 23–33 (1997)
22. Grosskreutz, H.: Probabilistic projection and belief update in the pGOLOG framework. In: Proceedings of the 2nd International Cognitive Robotics Workshop (CogRob'00) at the 14th European Conference on Artificial Intelligence (ECAI), pp. 34–41 (2000)
23. Grosskreutz, H., Lakemeyer, G.: On-line execution of cc-Golog plans. In: B. Nebel (ed.) Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), pp. 12–18. Morgan Kaufmann (2001)
24. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
25. Huang, X., Alleva, F., Hon, H.W., Hwang, M.Y., Rosenfeld, R.: The SPHINX-II speech recognition system: an overview. Computer Speech and Language **7**(2), 137–148 (1993)
26. Jacobs, S., Ferrein, A., Schiffer, S., Beck, D., Lakemeyer, G.: Robust collision avoidance in unknown domestic environments. In: Proceedings of the International RoboCup Symposium 2009 (RoboCup 2009), *LNCS*, vol. 5949, pp. 116–127. Springer (2009)
27. Ji, J., Chen, X.: Induction in nonmonotonic causal theories for a domestic service robot. In: Proceedings of the 21st International Conference on Inductive Logic Programming (ILP 2011) (2011)
28. Lemai, S., Ingrand, F.: Interleaving temporal planning and execution in robotics domains. In: AAAI'04:Proceedings of the 19th National Conference on Artifical Intelligence, pp. 617–622. AAAI Press / The MIT Press (2004)

29. Levesque, H., Reiter, R., Lésperance, Y., Lin, F., Scherl, R.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming **31**(1-3), 59–83 (1997)
30. Lienhart, R., Maydt, J.: An extended set of haar-like features for rapid object detection. In: Proceedings of the International Conference on Image Processing (ICIIP), pp. 900–903. IEEE, Rochester, USA (2002)
31. McCarthy, J.: Situations, Actions and Causal Laws. Tech. rep., Stanford University (1963)
32. Niemueller, T., Ferrein, A., Beck, D., Lakemeyer, G.: Design principles of the component-based robot software framework fawkes. In: Second International Conference on Simulation, Modeling, and Programming for Autonomous Robots. Springer, Springer, Darmstadt, Germany (2010)
33. Reiter, R.: Knowledge in Action. MIT Press (2001)
34. Schiffer, S., Baumgartner, T., Lakemeyer, G.: A modular approach to gesture recognition for interaction with a domestic service robot. In: S. Jeschke, H. Liu, D. Schilberg (eds.) Intelligent Robotics and Applications, *Lecture Notes in Computer Science*, vol. 7102, pp. 348–357. Springer Berlin / Heidelberg (2011)
35. Schiffer, S., Ferrein, A., Lakemeyer, G.: Reasoning with qualitative positional information for domestic domains in the situation calculus. Journal of Intelligent & Robotic Systems pp. 273–300
36. Schiffer, S., Ferrein, A., Lakemeyer, G.: Football is coming home. In: Proceedings of the 2006 International Symposium on Practical Cognitive Agents and Robots (PCAR'06), pp. 39–50. ACM, New York, NY, USA (2006)
37. Schiffer, S., Hoppe, N., Lakemyer, G.: Flexible command interpretation on an interactive domestic service robot. In: J. Filipe, A. Fred (eds.) Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), vol. 1 - Artificial Intelligence, pp. 26–35. SciTePress (2012). Best Student Paper Award
38. Schiffer, S., Wortmann, A., Lakemeyer, G.: Self-Maintenance for Autonomous Robots controlled by ReadyLog. In: F. Ingrand, J. Guiochet (eds.) Proceedings of the 7th IARP Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE'2010), pp. 101–107. Toulouse, France (2010)
39. Shibata, T., Mitsui, T., Wada, K., Tanie, K.: Entertainment and amusement robot technologies. subjective evaluation of seal robot: Paro. tabulation and analysis of questionnaire results. J Rob Mechatron **14**(1), 13–19 (2002)
40. Strack, A., Ferrein, A., Lakemeyer, G.: Laser-Based Localization with Sparse Landmarks. In: A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (eds.) RoboCup 2005: Robot Soccer World Cup IX, *Lecture Notes in Computer Science*, vol. 4020, pp. 569–576. Springer (2006)
41. Stückler, J., Behnke, S.: Integrating indoor mobility, object manipulation, and intuitive interaction for domestic service tasks. In: Proceedings of the 9th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2009), pp. 506 –513 (2009)
42. Thrun, S.: 175 miles through the desert. In: Proceedings of the 28th annual German conference on Advances in Artificial Intelligence, KI'05, pp. 17–17. Springer-Verlag, Berlin, Heidelberg (2005)
43. Viola, P.A., Jones, M.J.: Rapid object detection using a boosted cascade of simple features. In: 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), pp. 511–518 (2001)
44. Wisspeintner, T., van der Zant, T., Iocchi, L., Schiffer, S.: Robocup@home: Scientific Competition and Benchmarking for Domestic Service Robots. Interaction Studies. Special Issue on Robots in the Wild **10**(3), 392–426 (2009)
45. Wobbrock, J.O., Wilson, A.D., Li, Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In: Proceedings of the 20th annual ACM symposium on User interface software and technology, pp. 159–168 (2007)
46. van der Zant, T., Wisspeintner, T.: RoboCup X: A proposal for a new league where RoboCup goes real world. In: A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (eds.) RoboCup 2005: Robot Soccer World Cup IX, *Lecture Notes in Computer Science*, vol. 4020, pp. 166–172. Springer (2006)
47. van der Zant, T., Wisspeintner, T.: RoboCup@Home: Creating and Benchmarking Tomorrows Service Robot Applications. In: P. Lima (ed.) Robotic Soccer, pp. 521–528. InTech Education and Publishing (2007)