# Planning with Task-oriented Knowledge Acquisition for A Service Robot

**Kai Chen**[1], **Fangkai Yang**[2] and **Xiaoping Chen**[3*]
[1,3]School of Computer Science and Technology, University of Science and Technology of China
96 Jinzhai Rd, Hefei, Anhui, 200027, China
[1]chk0105@mail.ustc.edu.cn, [3]xpchen@ustc.edu.cn
[2] Katy Drilling Software Center, Schlumberger Software Technology, Schlumberger Ltd.
23500 Colonial Parkway, Katy, TX, 77493, USA
fyang10@slb.com

## Abstract

We propose a framework for a service robot to behave intelligently in domains that contain incomplete information, underspecified goals and dynamic change. Human robot interaction (HRI), sensing actions and physical actions are uniformly formalized in action language $\mathcal{BC}$. An answer set solver is called to generate plans that guide the robot to acquire task-oriented knowledge and execute actions to achieve its goal, including interacting with human to gather information and sensing the environment to help motion planning. By continuously interpreting and grounding useful sensing information, robot is able to use contingent knowledge to adapt to unexpected changes and faults. We evaluate the approach on service robot Kejia that serves drink to guests, a testing benchmark for general purpose service robot proposed by Robocup@Home competition.

## 1 Introduction

Research on domestic service robots has received increasing attention in recent years. A domestic service robot scenario combines the research on autonomous robots, human-robot interaction (HRI), computer vision, motion control and automated planning. Typical tasks of a domestic service robot include taking orders and serving drinks, welcoming and guiding guests, or just cleaning up [Wachsmuth *et al.*, 2015]. To measure and compare the performance of such robotics system, starting 2006, Robocup@Home, a part of Robocup initiative (www.robocup.org), defines a series of tests and one of the most challenging tests is General Purpose Service Robot (GPSR). An operator verbally specifies a complex, usually partially defined task to the robot that may request any skill, for instance, "serve Pepsi to Alice". The robot, which only has brief knowledge about the domain, needs to perform it, report any problems, adapt to unexpected changes and find alternative solutions.

Automated planning has been widely used for task planning and control in many service robot applications. In this paper we are interested in developing a service robot whose task planning integrates task-oriented knowledge acquisition. We believe acquiring task-relevant knowledge proactively in the context of specific planning problem will lead to a more robust service robot facing incomplete domain, uncertainty and faults. We particularly focus on three kinds of knowledge. (i) Domain knowledge. The robot only has limited amount of information about the domain or receives underspecified goal. In order to perform its task to serve Pepsi whose location is unknown, the robot needs to figure out that "location" is a piece of missing information (instead of "color", for example, in this specific plan), and acquires it by asking human. (ii) Control knowledge. When the robot is facing a dining table and about to pick up Pepsi, it will measure the distance of the object to determine if it should move closer, adjust its gripper, or simply fetch. The robot needs to figure out at planning time that the distance of objects may affect its manipulation strategy. (iii) Contingent knowledge. Throughout performing the task, the robot should continuously observe the environment, gather useful information, enrich its knowledge and adapt to the change. This is particularly important because the objects in domestic environment keep changing and the information provided by human can be vague or erroneous. Consequently, the robot has to start from a partial, incomplete and unreliable domain representation to generate plans to gather more information, in order to achieve its goal.

We propose a uniform way to formalize HRI, sensing and the physical actions together in a formal representation and excute plans using classical "plan-execute-monitor-replan" loop. Our method features: (a) a general modeling methodology of using $\mathcal{BC}$ [Lee *et al.*, 2013] to represent the world state, the robot's belief state and how they are affected by HRI actions, sensing actions and physical actions; (b) a continuous observation mechanism to the execution loop to acquire contingent knowledge; (c) a mechanism of symbol grounding that captures objects whose properties dynamically chage in the environment. The framework is implemented and tested in robot Kejia [Chen *et al.*, 2014], the champion of Robocup@Home competition in 2014 and 2nd in 2015, who serves drinks in a home environment.

## 2 Framework Overview

Shown in Figure 1, *domain formalization* in $\mathcal{BC}$ represents type hierarchy, domain objects and causal laws. The causal laws formalize effects of physical actions, HRI actions and sensing actions on fluents that represent world state and the

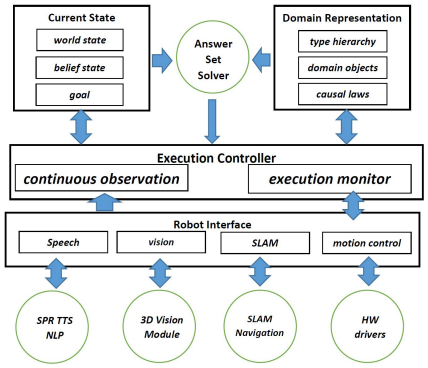


**Figure 1:** The architecture overview

robot's belief state. Initial domain objects are minimal and are enriched when new information is discovered. Human operator verbally specifies a goal which is translated by the semantic parser into the goal representation. The robot uses its sensor to obtain available world state and initialize its belief state. They constitute of the initial *current state* of a planning problem. After that, the planner, which is actually an *answer set solver*, is called to generate a plan that consists of a sequence of actions, and a sequence of states before and after the execution of each action.

Actions and states are sent to the *execution controller*. *Execution monitor* and *continuous observation* start simultaneously. Continuous observation constantly receives inputs from *robot interface* that senses domain information, grounds sensing inputs into symbols and updates current state. If update happens, it triggers a background verification process to validate current plan, by calling answer set solver to perform temporal projection reasoning, and sets replan flag if validation fails. Execution monitor sends action commands to controllers in robot interface too, which operates hardware to perform actions and returns grounded HRI, physical, and sensing results. Execution monitor compares the results with the expected state. If discrepancy is detected, it calls for replan.

Action controllers in robot interface directly operate low-level functional modules. Functional modules generally represent domain information in numerical states, and symbolic states are obtained through symbol grounding policies. It is crucial to maintain a correct mapping between low-level numerical states and high-level symbolic states for the robot to behave consistently and correctly.

## 3 Domain Formalization and Plan Execution

We present a general methodology and examples of representing a domestic environment with rooms, furniture, drink and people in which a service robot serves people drinks, a typical GPSR testing domain in RoboCup competitions. Initially Alice may request service by saying "bring me Pepsi". The robot has no knowledge about the object locations, so it may ask "where is the Pepsi". Alice may tell the robot a specific location "on the dining table", a vague location "in the dining room", an erroneous location "on the cabinet", or simply say "I don't know". The robot utilizes this information to find Pepsi, grasp it, put it in the basket, and bring it to Alice. The robot needs to adapt to changing environment and recover from various software and hardware failures.

### 3.1 Formalization in $\mathcal{BC}$

Domain knowledge consists of (i) types, relations, objects and (ii) causal laws that formalize the transition system. Causal laws are divided into *static laws*, which describe how the value of a fluent is determined by other fluents, and *dynamic laws*, which describe how the values of the fluents change from one state to another.

**Types.** We use a rigid (time-independent) fluent to denote type and object membership. For instance, we denote *pepsi1* is a member of type *pepsi* by *pepsi(pepsi1)*. Type hierarchy and relations are formalized using static laws such as $obj(X)$ **if** $pepsi(X)$.

**Causal Laws.** Non-rigid fluents consists of two groups:

- *world state* is a collection of fluents that represent the properties of the current world. It is further divided into *definite world state* and *possible world state*. Definite world state is affected by physical actions, such as moving affects the location of the robot. Possible world state represents possible value of an unknown property of an object. Possible world fluents are justified by belief fluents when actual HRI and sensing occur.
- *belief state* consists of fluents that represent the robot's belief. For instance, the robot knows a bottle of coke is located in the kitchen, or the robot knows an object is within reachable distance to grasp. Belief state is affected directly by HRI actions and sensing actions. It is also updated by continuous observation.

The combination of fluents can be used to represent various "epistemic states". For instance, {*objloc(pepsi1,kitchen), objloc(pepsi1,diningroom), -believeobjloc(pepsi1,R)*} for all rooms $R$ means it is possible for *pepsi1* to be in kitchen or in dining room, but the robot does not know it. They are important for symbol grounding during continuous observation and execution (Section 5 and Section 6). Due to space limit, below we give examples of formalizing HRI actions and sensing actions, as formalizing physical actions using action languages has been well studied [Khandelwal *et al.*, 2014].

| | |
|---|---|
| **Fluents** | Definite World State: *roboloc*, *ordertaken*, *gripperempty*, *lifted*, *lifterheight*<br>Possible World State: *objloc*, *at*, *distance*<br>Belief State: *believeobjloc*,*believeplat*, *believedistance*,*needsearch* |
| **Actions** | Physical Actions: *askorder*,*moveto*,*fetch*, *adjustgotopose*,*adjustpose*,*setlifterheight*,*fetch*, *liftup*, *stepback*,*putinbasket*,*handover*<br>HRI Actions: *askobjloc*<br>Sensing Actions: *detplane*,*checkdistance* |

**Figure 2:** List of non-rigid fluents and actions

The robot asks human the location of an object. To formalize *askobjloc*, a possible world fluent *objloc(O,L)* denotes the object $O$ is possible to be at location $L$ and a belief state fluent *believeobjloc(O,L)* that denotes the "robot believes that"

object $O$ is located at location $L$. When facing the person $P$, by executing action *askbojloc(O,P)* that asks person $P$ the location of $O$, the robot can obtain *believeobjloc(O,L)* if *objloc(O,L)* is possible. It is formalized by

$$\textit{askobjloc}(O, P) \textbf{ causes } \textit{believeobjloc}(O, L) \textbf{ if } \textit{roboloc}(P), \textit{objloc}(O, L).$$

When the actual location is not given, there are many possibilities about the location of an object. Consequently, the above causal law in practice simulates conditional effect of an action. Formally studying the modal properties of this particular representation pattern using $\mathcal{BC}$ is, however, beyond the scope of this paper and is a part of our future work.

During object manipulation, the robot approaches furniture and detects planes. Known plane heights of different furniture allows the robot to adjust its camera to proper angle for more accurate object recognition. This action is denoted by $\textit{detplane}(Pl, F)$. We use a pair of fluents: $\textit{at}(Pl, F)$, a possible world fluent that denotes plane *Pl* is possibly on furniture $F$, and $\textit{believeplat}(Pl, F)$, a belief state fluent that denotes the robot believes plane *Pl* is on furniture $F$:

$$\textit{detplane}(Pl, F) \textbf{ causes } \textit{believeplat}(Pl, F) \textbf{ if } \textit{at}(Pl, F), \textit{robotloc}(F), \textit{believeobjloc}(O, F).$$

The second sensing action for manipulation is *checkdistance(O,F)*. It rotates the camera, identifies and localizes the object, then calculates the distance relative to the gripper. The distance can be grounded to *prefech* (the object can be directly grasped), *distadjust* (the robot needs to move closer) and *distgoto* (the robot needs to move to another side of the furniture to grasp). This action is formalized similarly by a pair of possible world fluent and belief fluent:

$$\textit{checkdistance}(O, F) \textbf{ causes } \textit{believedistance}(O, D) \textbf{ if } \textit{roboloc}(F), \textit{lifterheight}(Pl), \textit{believeat}(Pl, F), \textit{distance}(O, D).$$

Based on the distance, the robot can make adjustments (actions *adjustpose* to move closer, *adjustgotopose* to move to another side of the furniture), or fetch the object (*fetch*). Besides, we also have static laws that derive fluent values from other fluent values. They may be recursive in their nature:

$$\begin{array}{l} \textit{believeobjloc}(O, R) \textbf{ if } \textit{believeobjloc}(O, F), \textit{in}(F, R). \\ \textit{believeobjloc}(O, P) \textbf{ if } \textit{believeobjloc}(O, \textit{robot}), \textit{roboloc}(P). \end{array} \quad (1)$$

### 3.2 Plan Generation and Execution Controller

The main control loop for plan generation and execution follows the traditional "planning-execution-monitor-replan" loop. The initial fluent set $S$ for planning is generated as follows: (i) fluents that belong to definite world state are initialized based on robot's own sensor inputs; (ii) fluents that belong to belief state are initialized as negated literals, denoting that the robot does not know anything about them. Possible world state is not specified for initial state. According to the semantics of $\mathcal{BC}$, this will lead to all possible worlds to be generated for the initial state that complement the incomplete information. After that, the robot goes into a loop that keeps taking orders and serving people. The robot starts by identifying the person and asking "What can I do for you, Alice?" Human may reply: "Please bring me Pepsi." This sentence is processed by the semantic parser. The semantic parser is built upon a categorical combinatorial grammar parser [Xie *et al.*, 2013] that first translated natural language instructions into ASP rules. In this case, rules

$$\begin{array}{ll} \textit{goal(1)} & \leftarrow \textit{obtained}(O, \textit{alice}, \textit{maxstep}), \textit{pepsi}(O). \\ & \leftarrow \textit{not goal(1)}. \end{array} \quad (2)$$

will be added to goal representation. After a goal is generated, answer set solver is called to generate answer sets using the union of domain representation $D$, goal $G$ and initial fluent set $S$. In the returned answer set, a sequence of *actions* and *estates* that denote expected states before and after execution of actions are obtained. The current sequence of *actions* is sent to execution. The sequence of actions may successfully reach the goal, or need replan. For some unrecoverable failure or change, the goal may be aborted. The details of action execution and result updateis explained in Section 5.

## 4 Continuous Sensing

Continuous sensing is a mechanism that whenever the sensing modules discover new information, they will ground results and update current state $S$. It allows the robot to reduce domain uncertainty at the same time of performing actions. As a result, the robot can be a lot more adaptive and robust to a changing domain and unreliable actions.

Continuous observation constantly monitors fluents *gripperempty*, *believeplatloc*, *believeobjloc* and *believedistance*. Vision module uses 3D Kinect cameras to constantly recognize planes on the furniture and add facts to $S$. For instance, if a new plane is detected from dining table, a new symbol *diningtableplane1* is generated, and *believeplat(diningtableplane1,diningtable1)* is added to $S$. At the same time, the numerical values denoting the height and location of the plane are also stored. The next time when a plane at similar location is detected, the robot will compare the location with the previous one, and recognize it as *diningtableplane1* to avoid duplicates. Similarly, the vision module recognizes different kinds of drinks and their locations, generates symbols and adds grounded facts using fluent *believeobjloc* to update $S$. Once the object is identified, its distance may be calculated. Based on a predefined interval, the calculated distance is classified into *prefetch*, *distgoto*, and *distadjust*. The symbolic distance information is added to $S$ using fluent *believedistance*. However, there are two extra cases to handle. The first case is when an object (e.g. *pepsi1*) is identified on the dining table but not within reach for the robot. According to our semantics of fluents, this result maps to fluents $\{\neg \textit{believedistance}(pepsi1, D), \neg \textit{distance}(pepsi1, D)\}$ where $D$ be *prefetch*, *adjustdist* or *adjustgoto*. Note that $\{\neg \textit{believedistance}(pepsi1, D)\}$ is not sufficient to represent this state, becaue it means the distance of *pepsi* is unknown (this is how we set the initial state, see Section 3.2). The second case is when vision module does not recognize any *pepsi1* when the robot is looking for it. In addition to setting the above fluents, $\textit{believeobjloc}(\textit{pepsi1}, \textit{diningtable1})$ is set to *false* as well. This negative literal indicates the robot abandons the information *believeobjloc(pepsi1,diningtable1)* that is obtained from human or continuous sensing. New plan may involve searching from nearby places or asking person again.

These facts enrich the knowledge about the domain and help the robot to find better plans. It should be noted that object recognition algorithm used by the vision module can more reliably identify objects if the robot is not moving. So if the robot believes an object is at certain location, while continuous observation does not recognize it, it will not update the robot's belief by removing *believeobjloc* fluent for this object, because it may be a false positive.

Fluent values may be derived from other fluents by static laws such as (1). When continuous observation sets *believeobjloc(pepsi1,diningtable1)* to be true, *believeobjloc(pepsi1,diningroom1)* needs to be added to $S$ too. It can be derived by *cautious reasoning* with *believeobjloc(pepsi1,diningtable1)* and static laws of $D$ using answer set solver (Line 5). *Cautious consequence* of an answer set program is the intersection of all answer sets. In some answer set solver such as CLASP [Gebser *et al.*, 2012], it is supported by using a command-line option `-e cautious`.

Every time when $S$ is updated, robot verifies if the current plan is still the best one in presence of the new information. A previous plan may easily become invalid, because, for instance, an object to retrieve disappears unexpectedly or the gripper does not catch the object. To validate the current plan, we perform a *temporal projection reasoning* [Lee *et al.*, 2013, Section 7] by projecting the end result of executing the remaining actions of the current plan in presence of the new information. If the goal is satisfied in the projected result and a shorter plan is not found, it means the current plan is still the best. Otherwise, a flag indicator for replan is set, and the current plan is aborted.

## 5 Execution Monitor and Action Controllers

Execution monitor dispatches action commands to corresponding controllers in robot interface for execution. Controllers operate hardware and, like in continuous observation, ground and return symbolic results. The controllers send low-level control commands to corresponding functional modules (Line 6). In case of *fetch*, motion planner generates motion trajectories. For HRI action *askobjloc*, speech module synthesizes voice and speaks to a person. For sensing action *checkdistance*, the controller is the same used in continuous observation. The current action may be aborted due to continuous sensing.

When action execution finishes, its full effect is derived through symbol grounding and cautious reasoning. Symbol grounding is controller-specific. Manipulation tasks such as *fetch* is successful when the arm and gripper successfully move along all trajectories and pressure sensor indicates an object is grasped. For HRI action *askobjloc*, it may be grounded to atoms like *believeobjloc(pepsi1,diningroom1)*) for vague information, *believeobjloc(pepsi1,diningtable1)*) for precise information, or *needsearch(pepsi1)* when human answers "I don't know". Then the robot can search every furniture for Pepsi. Symbol grounding for sensing actions is handled the same way as in continuous observation. The execution result is compared with expected state to determine if replan is needed or goal is achieved.

## 6 Experiment and Evaluation

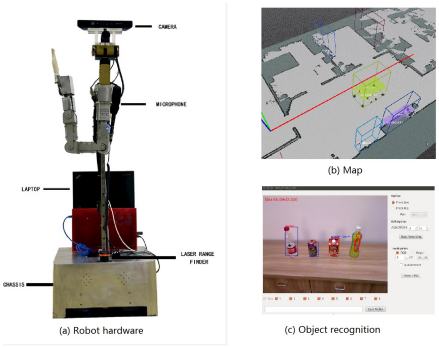


**Figure 3:** The robot hardware, SLAM generated map and vision

The presented framework is implemented in Kejia. Kejia (Figure 3(a)) is equipped with a wheeled mobile base, a single 5-Degree-of-Freedom arm, a microphone, 2D laser range finder, Microsoft Kinect and a high-resolution fire wire. Lower level modules includes motion control which drives the mobile base and arm, 2D SLAM and navigation module (Figure 3(b)), 3D vision to recognize and localize furniture planes and pre-trained objects such as water, Pepsi, vitamin drink and apple juice (Figure 3(c)). Our system is implemented as a node in ROS (Robot Operating System) network. It subscribes all necessary information (hardware feedbacks, robot pose, recognized objects, etc.) and publishes control messages (navigate to some location, turn on object recognition function, etc.) which affect the behavior of the robot.

Such robot is constantly affected by uncertainties: hardware malfunction (e.g. arm stuck during grasping), perception errors (e.g. error in localization and recognized object positions, false positives and low recognition rates), HRI uncertainties (erroneous/vague information from human) and changing environment (recognized object removed by human). Since initially drinks and their locations are unknown to the robot and there are so many changing factors, it is extremely important for the robot to constantly gather domain information using HRI and sensing, adapt to change and recover from failure. In experiment we use CLASP as our answer set solver, and serving one person typically need the robot to execute 9–12 actions. Our demo video (`https://youtu.be/zoKNFozlFPk`) shows 7 consecutive tasks in one trial how robot responds to so many challenges.

**Scenario 1.** Alice requested water. The robot started by asking Alice the location. Alice offered vague information: in the dining room. The robot started searching in dining room by visiting the end table first. On the end table, there were Pepsi and water. However, due to unreliable vision, the robot recognized them to be water, and brought one to Alice.

**Scenario 2.** Bob requested Pepsi. Since the robot didn't recognize Pepsi in the previous task, it asked Bob the location and obtained a correct answer: "end table". At this very moment, Alice came to move the coke from end table to the dining table, making Bob's information erroneous. The robot, after failed to identify the Pepsi on the end table, abandoned


**Figure 4:** Results of Experiment

the information from Bob, and found Pepsi on the side table. However, due to manipulation error, the Pepsi slipped out from the basket.

**Scenario 3.** Alice requested Vitamin. The robot got the answer "dining table" from Alice and moved to the dining table, but unfortunately, Vitamin was too far away for the robot to recognize. However, the robot recognized closer objects: water, Pepsi and apple juice. Consequently, the robot abandoned human information and searched by itself. Finally, Vitamin was found on the side board and was delivered to Alice.

**Scenario 4.** Bob requested water. The robot directly moved to the end table because in Scenario 1, it mistakenly identified Pepsi as water there. This time, vision correctly identified that water was not there, and the robot found an alternative by moving to the dining table and grasped water identified in Scenario 3.

**Scenario 5.** Alice requested apple juice. The robot directly moved to dining table and grasped the apple juice identified in Scenario 3. Vitamin was also recognized this time.

**Scenario 6.** Bob requested Vitamin. The robot moved to dining table for Vitamin directly. After measuring distance, it adjusted its position by moving to another position to grasp. The first attempt failed and was captured by continuous sensing immediately. A new plan was generated and executed, which was successful.

**Scenario 7.** Alice requested Pepsi. The robot directly moved to Pepsi and grasped it. Unfortunately, the gripper hit the can and stuck in its trajectory, leading to manipulation failure.

Despite so many uncertainties and failures, the robot managed to handle most of them pretty robustly and achieved 5 of 7 tasks, by adaptive planning and reasoning. Vision errors tend to occur more frequently in the early stage, but they are all corrected later. Furthermore, after the 3rd run, the robot no longer asked human questions because it has already acquired sufficient information about drinks and their locations.

Following the scoring policy of Robocup@Home competition, we evaluate our framework by conducting a total of over 20 hours running the robot and finished about 157 orders in 45 trials. The trial stops due to unrecoverable fault. As is shown in Figure 4, most often the robot can successfully serves 2–4 persons consecutively, with the average being 3 persons. Despite that, there are two trials that the robot managed to take 8 orders. Given the challenge of GPSR domain, the experiment demonstrates robustness and the ability to tackle uncertainty, failure and incomplete information.

## 7 Conclusion

In this paper we proposed a framework of building general purpose service robot, by handling HRI, sensing and physical actions in a uniform representation and execution loop with continuous observation. Our experiment demonstrates robustness of the service robot in domestic environment. We conclude that, proper use of symbolic planning combined with task-oriented knowledge acquisition can be helpful to handle unpredictable domain changes and perception errors, two challenges in all GPSR domains. In the future, we will address more complex domains with more reliable control.

## References

[Chen *et al.*, 2014] Kai Chen, Dongcai Lu, Yingfeng Chen, Keke Tang, Ningyang Wang, and Xiaoping Chen. The intelligent techniques in robot kejia–the champion of robocup@ home 2014. In *RoboCup 2014: Robot World Cup XVIII*, pages 130–141. Springer, 2014.

[Gebser *et al.*, 2012] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, 2012.

[Khandelwal *et al.*, 2014] Piyush Khandelwal, Fangkai Yang, Matteo Leonetti, Vladimir Lifschitz, and Peter Stone. Planning in Action Language $\mathcal{BC}$ while Learning Action Costs for Mobile Robots. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

[Lee *et al.*, 2013] Joohyung Lee, Vladimir Lifschitz, and Fangkai Yang. Action Language $\mathcal{BC}$: A Preliminary Report. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Wachsmuth *et al.*, 2015] Sven Wachsmuth, Dirk Holz, Maja Rudinac, and Javier Ruiz-del Solar. Robocup@home – benchmarking domestic service robots. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Xie *et al.*, 2013] Jiongkun Xie, Xiaoping Chen, and Zhiqiang Sui. Translating action knowledge into high-level semantic representations for cognitive robots. *Nonmonotonic Reasoning, Action and Change*, page 53, 2013.