An Agent-Based Cognitive Robot Architecture

Changyun Wei and Koen V. Hindriks

Interactive Intelligence, Delft University of Technology, The Netherlands
{C.Wei,K.V.Hindriks}@tudelft.nl

Abstract. We propose a new *cognitive robot control architecture* in which the cognitive layer can be *programmed* by means of the agent programming language Goal. The architecture exploits the support that agent-oriented programming offers for creating *cognitive robotic agents*, including symbolic knowledge representation, deliberation via modular, high-level action selection, and support for multiple, declarative goals. The benefits of the architecture are that it provides a flexible approach to develop cognitive robots and support for a clean and clear separation of concerns about symbolic reasoning and sub-symbolic processing. We discuss the design of our architecture and discuss the issue of translating sub-symbolic information and behavior control into symbolic representations needed at the cognitive layer. An interactive navigation task is presented as a proof of concept.

1 Introduction

The main motivation for our work is the need for a flexible, generic, high-level control framework that facilitates the development of re-taskable robot systems and provides a feasible alternative to the usual task- and domain-dependent development of high-level robot control. As cognitive robots are supposed to handle complex reasoning problems in dynamic environments [1], we believe that agent-oriented programming offers such an approach as it supports the programming of cognitive agents. Using agent programs to create the cognitive layer in a robot control architecture is natural and provides several benefits. It becomes relatively easy to adapt the control *at the cognitive layer* itself to various domains. This approach is flexible and, if functionality of other layers is generic and can be used in multiple task domains, facilitates reuse. An agent-based approach, moreover, provides support for autonomous, reactive, and proactive behaviors and also endows a robot with the required deliberation mechanism to decide what to do next [2]. Of course, generality may come at a trade-off and does not imply that a generic architecture will always perform better than a dedicated robot control architecture [3].

Designing and developing a *cognitive robot control architecture* poses several challenges. Robots are embedded systems that operate in physical, dynamic environments and need to be capable of operating in real-time. A range of perception and motor control activities need to be integrated into the architecture. This poses a challenge for a cognitive, symbolic architecture as "it can be particularly difficult to generate meaningful symbols for the symbolic components of

M. Dastani, J.F. Hübner, and B. Logan (Eds.): ProMAS 2012, LNAI 7837, pp. 54–71, 2013.

cognitive architectures to reason about from (potentially noisy) sensor data or to perform some low-level tasks such as control of motors" [4]. Ideally, moreover, such an architecture should provide support for the integration or exchange of new and different sensors and behaviors when needed. Given the complexity and the number of components needed in a robot control architecture, one also needs to consider how all the processing components in the system communicate and interact with each other [5].

The main focus and contribution of this paper is to provide a simple but also generic and flexible solution to integrate the knowledge representation and reasoning capabilities needed for a cognitive robot [6, 7]. As perception and action need to be tightly coupled for a cognitive robot to be able to operate effectively [1], we also discuss this relationship. We propose an agent-based cognitive robot control architecture that integrates low-level *sub-symbolic* control with high-level *symbolic* control into the robot control framework. We use the agent programming language GOAL [8, 9] for implementing the cognitive layer, whereas low-level execution control and processing of sensor data are delegated to components in other layers in the proposed architecture. GOAL, among others, supports goal-oriented behavior and the decomposition of complex behavior by means of modules that can focus their attention on relevant sub-goals. GOAL has already been successfully applied to control real-time, dynamic environments [10], and here we demonstrate that it also provides a feasible approach for controlling robots. In our approach, the cognitive layer is cleanly separated from the other layers by using the Environment Interface Standard (EIS; [11]). As a proof of concept, we will use a task, in which a robot will navigate in an office environment in order to deliver a message to one of our colleagues, to show how the agent-based cognitive control can be realized in physical robots. The main contribution of our proposed architecture is that it provides:

- a decoupled framework for combining low-level behavioral robot control with high-level cognitive reasoning,
- a generic interface for mapping sensory information to symbolic representations needed at the cognitive layer, and
- a flexible mechanism in the cognitive layer for synchronizing percepts and actions.

The paper is structured as follows. Section 2 briefly discusses some related work. Section 3 presents and discusses the design of the cognitive robot control architecture. Section 4 presents a proof of concept implementation. Section 5 concludes the paper and discusses future work.

2 Related Work

Cognitive robots are *autonomous* and *intelligent* robot systems that can perform tasks in real world environments without any external control, and are able to make decisions and select actions in dynamic environments [12]. It remains a challenge, however, to integrate the symbolic problem solving techniques such

as knowledge representation, reasoning, and planning developed within artificial intelligence with sub-symbolic functions such as perception and motor control on a robot [13]. It has been argued, however, that knowledge representation is important to enable robots to reason about its environment [1, 6, 7]. Here we briefly discuss some related work that either explicitly aims at developing a cognitive robot architecture or uses some kind of symbolic representation for controlling a robot.

The work that is most similar in spirit to our own is that of [14] and [15]. In [14] the high-level language Golog is used for controlling a robot. Golog supports writing control programs in a high-level, logical language, and provides an interpreter that, given a logical axiomatization of a domain, will determine a plan. [15] proposes teleo-reactive programming for controlling a robot. A teleo-reactive program consists of multiple, prioritized condition-action rules. However, neither Golog nor teleo-reactive programming provide a BDI perspective on programming agents. Moreover, these papers do not discuss the robot control architecture that allows the symbolic framework to control the robot.

CRAM [2] is a software toolbox designed for controlling the Rosie robot platform developed at the Technische Universität München. It makes use of Prolog and includes a plan language that provides a construct for concurrent actions. The CRAM approach also aims at providing a flexible alternative to pre-programmed robot control programs. [16] proposes to use a common sense ontology for defining predicates for high-level control of a robot that is integrated into the CRAM architecture. One difference between the CRAM and our approach is that the reasoning and planning components are two separate components in CRAM whereas they are integrated into an agent architecture in our robot control framework. Moreover, we propose the use of an explicit interface component for connecting the low-level and high-level control in our architecture that allows for a clean separation of concerns.

It has been argued that building robot systems for environments in which robots need to co-exist and cooperate with humans requires taking a *cognitive stance* [17] . According to [17], translating the key issues that such robot systems have to deal with requires a cognitive robot control architecture. Taking a cognitive stance towards the design and implementation of a robot system means that such a system needs to be designed to perform a range of *cognitive functions*. Various *cognitive architectures*, such as ACT-R [18] and SOAR [19], have been used to control robots. These architectures were not primarily aimed at addressing the robot control problem and in this sense are similar to agent programming languages, the technology that we advocate here for controlling robots. SOAR has been used to control the hexapod HexCrawler and a wheeled robot called the SuperDroid [4]. ADAPT (Adaptive Dynamics and Active Perception for Thought) is a cognitive architecture based on SOAR that is specifically designed for robotics [20]. The SS-RICS (Symbolic and Sub-symbolic Robotic Intelligent Control System) architecture for controlling robots is based on ACT-R; SS-RICS is intended to be a theory of robotic cognition based on human cognition [21, 3]. This work mainly focuses on integrating a broad range of cognitive

capabilities into an architecture instead of on more pragmatic issues related to programming cognitive robotics and reuse. Unlike [22], we are not mainly concerned here with the long-term goal of developing robotic systems that have the full range of cognitive abilities of humans based on a cognitive architecture such as SOAR. Our work is oriented towards providing a more pragmatic solution for the robot control problem as discussed above. Still our work may contribute to the larger goal that [22] sets as there appear to be quite a few similarities between BDI-based agents and cognitive architectures such as ACT-R and SOAR).

3 Cognitive Robot Control Architecture

This section introduces our cognitive robot control architecture. We discuss several issues including the processing and mapping of sensory data to a symbolic representation, the translation of high-level decisions into low-level motor control commands, and the interaction of the various architecture components.

3.1 Overall Design of the Architecture

A high-level overview of our layered architecture is shown in Figure 1. The architecture consists of four layers including a symbolic, cognitive layer realized by means of the agent programming language GOAL, a middle layer for controlling robot behavior (written in C++), and a hardware control layer (using URBI[1], a robotic programming language). The Environment Interface Standard (EIS) layer provides the technology we have used to manage the interaction between the symbolic and sub-symbolic layers. We argue here that EIS provides a tool that can be extended to deal with the issue of translating *sub-symbolic* sensory data that consists typically of noisy, incomplete, and quantitative measurements into *symbolic representations* that are needed in the symbolic cognitive layers to support reasoning. Note that the environment layer is not part of the architecture but refers to the physical environment that the robot operates in.

The main functionality for controlling a robot is placed in the behavioral control layer. In addition to the functions such as (object) recognition, navigation, localization, path planning and other common functions, this layer is also responsible for communicating with the higher-level symbolic components, including the interpretation of symbolic messages that represent actions and making the robot perform these actions in its physical environment.

The interface layer acts as a bridge between the behavioral and cognitive control layers. Because these layers use different languages for representing subsymbolic and symbolic information, respectively, we need an interface to translate between these representations. The cognitive control layer acts as a task manager for the robot and provides support for managing the robot's *mental state* which allows the robot to keep track of what is happening while executing a task.

[1] http://www.urbiforge.org/

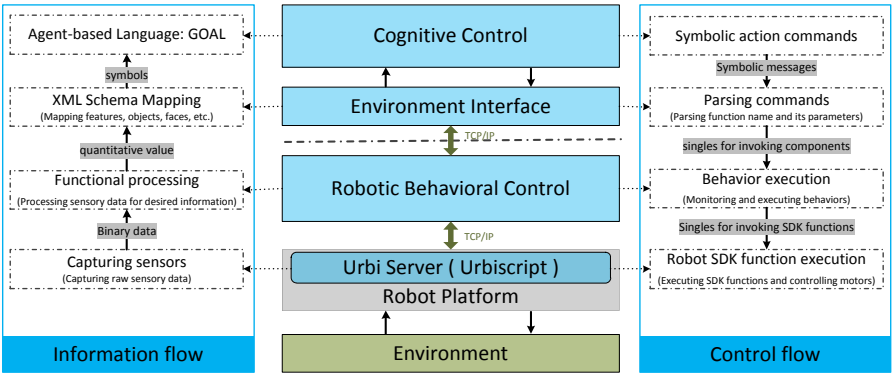


Fig. 1. The overall design of the architecture

3.2 System Architecture and Components

A robot control architecture provides an organizational structure for software components that control a robotic system [23]; such architectures are specialized because of the unique requirements that embedded systems such as robots impose on software. Here we will discuss the detailed architecture shown in Figure 2 that we propose that matches the layered architecture framework of Figure 1.

Robot Platform. The architecture has been implemented on the humanoid NAO robot platform from Aldebaran Robotics.[2] We have used the URBI middleware [24] that provides the *urbiscript* language for interfacing with the robot's hardware. We have chosen URBI instead of the Robot Operating System (ROS[3]) platform because it does not include the orchestration layers present in URBI that provide support for parallel, tag-based, and event-driven programming of behavior scripts. When the application is executed, an urbiscript program, which initializes all API parameters of sensors and motors (e.g., the resolution and frame rate of camera images), is sent to configure the robot platform.

Behavioral Control. The behavioral control layer is written in C++, connecting the robot hardware layer with higher layers via a TCP/IP connection. This layer is mainly responsible for information processing, knowledge processing, communication with the deliberative reasoning layer and external robots, and action and behavior executions. All of these components can operate concurrently. The main functional modules include:

– **Sensing**, for processing sensory data and receiving messages from other robots. Sensors that have been included are sonar, camera, microphone, an inertial sensor, and all sensors monitoring motors. Because the memory space

[2] http://www.aldebaran-robotics.com/
[3] http://www.ros.org/

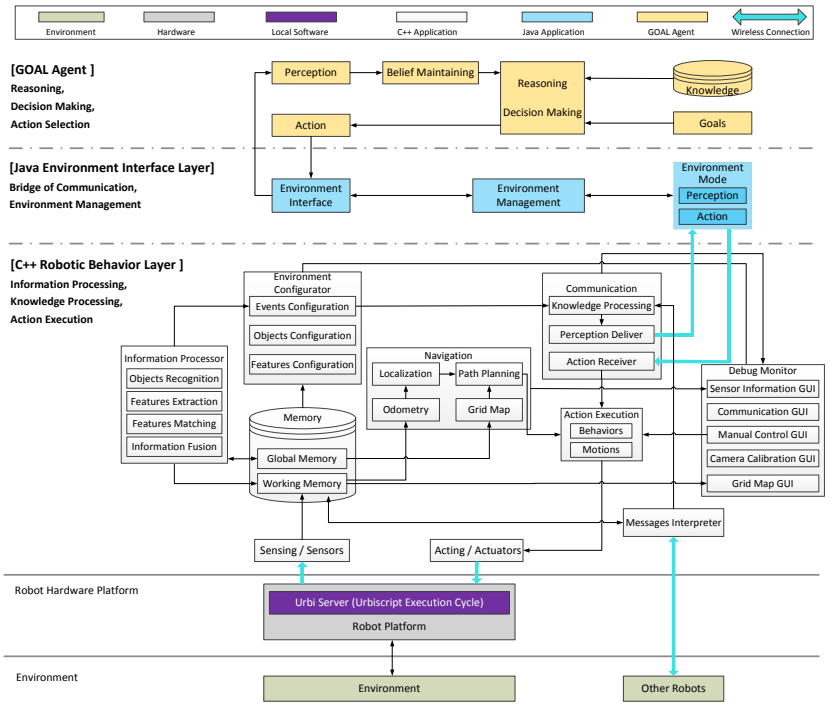


Fig. 2. Overview of the agent-based cognitive robot architecture

required for camera images is significantly bigger than that for other sensors, the transmission of images is realized via a separate communication channel.

– **Memory**, for maintaining the *global memory* and *working memory* for the behavioral layer. In global memory, a map of the environment and properties of objects or features extracted from sensor data are stored. We have used a 2D grid map for representing the environment. This map also keeps track which of the grid cells are occupied and which are available for path planning. The working memory stores temporary sensor data (e.g., images, sonar values) that are used for updating the global memory.
– **Information Processor**, for image processing. This component provides support for object recognition, feature extraction and matching, as well as for information fusion. Information fusion is used to generate more reliable information from the data received from different sensors. For example, the odometry navigation component is only able to provide a rough estimate of the robot's position due to joint backlash and foot slippage. To compensate for this, the robot has also been equipped with the capability to actively re-localize or correct its position by means of predefined landmarks. We have used the OpenCV [25] library to implement algorithms and methods for processing images captured by the camera. Algorithms such as

Harris-SIFT [26], RANSAC [27], and SVM [28] for object recognition and scene classification have been integrated into this module.

- **Environment Configurator**, for interpreting and classifying events that occur in the environment. For example, when the output of the left sonar exceeds a certain value, this module may send a corresponding event message that has been pre-configured. This is useful in case such readings have special meaning in a domain. In such cases, an event message may be used to indicate what is happening and which objects and features such as human faces and pictures have been detected. This is a key component in our approach for mapping sub-symbolic data to symbolic representations.
- **Navigation** includes support for *localization*, *odometry*, *path planning* and *mapping* components, which aid the robot in localizing itself and in planning an optimal path to a destination in a dynamic, real-time environment. Several transformations between layers are required for navigation. The cognitive layer uses a 2D Grid-map Coordinate System (GCS) which is also used by the the path planning component. When following a planned path, after each walking step, the odometry component is used for keeping track of the robot's actual position and for providing the real-time coordinates of the robot which are used for planning its next walking step. The odometry sensors, however, provide the robot's coordinates in a so-called World Coordinate System (WCS) that needs to be mapped to the robot's position in the 2D Grid-map Coordinate System (GCS) for path planning. At the lowest layer that executes the walking steps, moreover, the GCS position has to be transformed into the Local Coordinate System (LCS) that the robot platform uses for actual motor movements.
- **Communication**, which provides outputs from the Environment Configurator to the interface layer and receives action messages from this same layer. The communication component mainly is a technical component that acts as a Server/Client infrastructure and uses the TCP/IP protocol for actual message delivery. The behavior layer in the architecture initiates and starts up a unique server to which a cognitive layer can connect as a client (thus facilitating the swapping of control from one robot to another).
- **Debugger Monitor** provides several GUIs that are useful for debugging robot programs, enabling developers to visualize sensory data and allowing them to set specific function parameters. This component also includes a *Wizard of Oz interface* to conduct human-robot interaction experiments.
- **Action Execution**, for instructing a robot to perform concrete behaviors and actions. The actions include motion movements such as walking and turning around while the behaviors include predefined body gestures such as sitting down, standing up and raising the arms of a humanoid robot.

Environment Interface. The interface layer between the behavior and cognitive layers has been built using a software package called Environment Interface Standard (EIS; [11]). The core components in this layer are an *Environment Model* component that establishes a connection between the cognitive layer with

the behavioral control layer, an *Environment Management* component that initializes and manages the interface, and an *Environment Interface* component that provides the actual bridge between the cognitive and behavioral layer.

The benefit of using EIS for implementing the interface layer is that it already provides a well-defined and structured language for representing knowledge, the so-called Environment Interface Language (EIL). Moreover, various agent platforms support EIS which implies that our architecture may be reused to connect languages other than GOAL without much effort to the robot platforms that are supported by the architecture. We thus obtain immediately a clear, simple but sufficiently expressive target language for mapping sub-symbolic data maintained in the behavioral layer to symbolic percepts used in the cognitive layer. Because this language is already supported by GOAL, this allows us moreover to focus completely on the problem of mapping sub-symbolic data to the EIL language. We discuss a generic scheme to do this in Section 3.4 below.

Deliberative Reasoning and Decision Making. Knowledge representation and reasoning is an essential component for a cognitive robot that allows such a robot to keep track of what is happening in its environment and to make rational decisions to change that environment [7, 6]. The cognitive control layer provides support for reasoning and decision making. In our architecture, we employ the GOAL agent programming language for programming the high-level cognitive agent that controls the robot. GOAL is a language for programming logic-based, cognitive agents that use symbolic representations for their beliefs and goals from which they derive their choice of action. Due to space limitations, we do not describe GOAL agent programs in any detail here but refer the interested reader for more information to [8, 9]. In Section 4, we will use a navigation task to illustrate how a GOAL agent program can be used for deliberative reasoning and decision making for controlling a robot.

3.3 Decoupled Architecture Layers

Similar to most layered architectures, we also distinguish reactive and deliberative control in our architecture. In our architecture these layers are loosely coupled and connected through a separate interface layer. We have deliberately chosen for this setup. Alternatively, these layers could have been more tightly coupled. Tight coupling has some benefits such as a more robust integration and a reduced communication overhead which often leads to higher performance than loosely coupled systems [29]. These benefits are obtained by using a memory component that is shared and directly accessible by all other components. This also avoids the need to "duplicate" information at different layers, although one should note that also in a tightly coupled setup mappings between sub-symbolic and symbolic data are needed. The disadvantages of a more tightly coupled approach are an increased complexity of the architecture and a higher interdependence between architecture components. As a consequence, it becomes more difficult to extend such architectures in order to be able to handle a range of different tasks [29].

Our choice to opt for decoupled layers is motivated by our objective to design an architecture that is as flexible as possible. A benefit of decoupling is that it provides for a clean separation of concerns. Moreover, decoupling of these layers facilitates the more or less independent programming of high-level, cognitive agents that control a robot as well as of lower-level behavioral functions. An agent programmer, for example, does not need to spend time handling object recognition. Similarly, a behavior programmer who codes in C++ does not have to consider decision making nor even master the agent programming language used in the cognitive layer. The main challenge that needs to be faced in a decoupled approach is how to translate or transform sub-symbolic data to symbolic representations, an issue to which we turn next.

3.4 Knowledge Acquisition and Representation

We have designed a generic method for transforming sub-symbolic data to symbolic representations that can be used in the cognitive layer. The main function of the interface layer is to perform the actual transformation. The interface layer thus needs to be provided with the functionality to process all sub-symbolic data that is sent to this layer by the behavioral layer. Of course, the type of data that needs to be processed and transformed depends on the robot platform that is used and the sensors available on that platform. In our case, we have listed all the available sensors and associated functionality for processing raw sensor data in the first two columns of Table 1. The key issue is how to interpret and map the sub-symbolic data obtained through sensors to a symbolic representation that is useful for decision making in the cognitive layer. In essence, what we need in order to obtain a transformation is a coding scheme. Such a coding scheme should not only indicate *how* to map data to a particular representation but also allow to specify *when* such a translation should be performed.

The solution that we propose as a generic solution for creating a coding scheme for transforming data into symbols is to use a standard template. To this end, we have used XML to create such a standard message template with the following structure:

```
<?xml version="1.0"?>
<message_percept>
  <descriptor  sensor="sensors" function="name">
      <para1>parameter</para1>
      <para2>parameter</para2>
      <para3>parameter</para3>
      ...
  </descriptor>
</message_percept>
```

Every data item sent from the robotic behavioral layer to the interface layer uses the above XML template, where `sensor` indicates which sensor this data item is produced from, `function` refers to the name of the function used for processing the sensor data, and `para1`, `para2`, `para3`) are parameters of functions that may have various number of parameters.

The XML schema above is used by the interface layer for mapping data items to symbolic representations in a generic and flexible manner. For each sensor

and associated functions, an XML schema is stored in a database used by the interface layer. Table 1 shows how to process the data from several sensors and map them to corresponding symbolic representations.

Table 1. Knowledge acquisition and representation

Sensors	Processing	Acquisition and Mapping	Representation
Sonar value	sonarLeftVal	thr2 ≤ para1 < thr1	`obstacle(left)`
		... ≤ para1 < thr2	`collision(left)`
	sonarRightVal	thr2 ≤ para1 < thr1	`obstacle(right)`
		... ≤ para1 < thr2	`collision(right)`
Inertial value	placeEstimation	para1 = x, para2 = y [x, y] ∈ room	`in(room)`
		para1 = x, para2 = y [x, y] ∈ room.hallway	`inFrontOf(room)`
	relativePose	para1 = x, para2 = y	`position(x,y)`
	walkIsActive	para1 = "true"	`walking`
		para1 = "false"	`static`
Camera image	featureRecognition	para1 = feature	`feature(feature)`
	colorDetection	para1 = color	`color(color)`
	shapeDetection	para1 = shape	`shape(shape)`
	absolutePose	para1 = x, para2 = y	`position(x,y)`
	sceneClassification	para1 = "true"	`doorOpen`
		para1 = "false"	`doorClosed`

As the number of the parameters of processing functions are various in order to convey different messages, we will discuss several conditions with respect to the various parameters, illustrating how to obtain adequate coding schemes using the XML schema proposed.

Functions with Binary Parameter. Some functions, such as `walkIsActive`, `sceneClassification`, only have a binary parameter, namely "true" or "false" value. Suppose the robot needs to figure out if the door in front of it is open or closed. What the deliberative reasoning needs is the symbolic percept: `doorOpen` or `doorClosed`. In order to obtain this percept, the knowledge processing function: `sceneClassification` should analyze the images from the robot's camera. SVM can be chosen as a discriminative classifier for scene classification. As a statistical learning technique, SVM should also handle the uncertainty problem of sensory data, in which the state of the door (i.e., histogram of images) first has to be trained to obtain an appropriate threshold for classification. The The knowledge acquisition and mapping procedure in the interface layer uses the XML schema to examine the contents:

```
<descriptor sensor="Camera image" function="sceneClassification">
    <para1="true"><interpreter>"doorOpen"</interpreter> </para1
    <para1="false"><interpreter>"doorClosed"</interpreter></para1
</descriptor>
```

If the `para1` is "true", this data item can be interpreted as `doorOpen`; likewise, if it is "false", a `doorClosed` percept can be generated.

Functions with Threshold Parameter. Some functions, such as `sonarLeftVal`, `sonarRightVal`, have a threshold parameter. Typically, a parameter will be examined to see if it exceeds a particular threshold or not. But in many practical situation, one parameter should be compared with two or more different thresholds so as to match to corresponding categories. Taking the `sonarLeftVal` as an example in Table 1, the XML schema in this case can be expressed as:

```
<descriptor sensor="Sonar value" function="sonarLeftVal>
    <thr1=0.8><interpreter>"obstacle(left)"</interpreter> </thr1
    <thr2=0.5><interpreter>"collision(left)"</interpreter></thr2
</descriptor>
```

If the data item intends to represent `obstacle(left)`, the `para1` should satisfy: $0.5 \leq para1 < 0.8$; if it intends to represent `collision(left)` as we have not define the third threshold `thr3`, the `para1` only needs to satisfy: $para1 < 0.5$. Note that this function can have as many as categories if we continually add thresholds and define corresponding interpreters.

Functions with Argument Parameter. The functions, such as `relativePose`, `absolutePose` and `placeEstimation`, have argument parameters, which provide position information of 2D coordinates: x and y. For `relativePose` and `absolutePose`, the coordinate data do not need to be mapped to specific symbolic representation, but we still need to add symbolic predicate (i.e., `position()`) in this percept message so that the cognitive layer can understand what kind of percepts it belongs to.

However, The `placeEstimation` handles the topological localization problems for place estimation. When a robot knows its 2D coordinate in the world map, the robot can know its topological position (e.g., in room A or in front of room A). The XML database for this condition can be:

```
<descriptor sensor="Camera image" function="placeEstimation">
    <rect x1=0, y1=100, x2=100, y2=0   ><interpreter>"in(roomA)"</interpreter></rect
    <rect x1=0, y1=200, x2=100, y2=100 ><interpreter>"in(roomB)"</interpreter></rect
    <rect x1=0, y1=300, x2=100, y2=200 ><interpreter>"in(roomC)"</interpreter></rect
    ...
    <rect x1=100, y1=100, x2=200, y2=0   ><interpreter>"inFrontOf(roomA)"</interpreter></rect
    <rect x1=100, y1=200, x2=200, y2=100 ><interpreter>"inFrontOf(roomB)"</interpreter></rect
    <rect x1=100, y1=300, x2=200, y2=200 ><interpreter>"inFrontOf(roomC)"</interpreter></rect
    ...
</descriptor>
```

The attribute of the `rect` describes a rectangle region defined by its upper left and lower right coordinates. If the parameters of the function locate within a defined region, namely $(x1 \leq x \leq x2) \cup (y2 \leq y \leq y1)$. Once the environmental map is known, the topological places can be defined in detail using this XML schema.

Functions with Identification Parameter. Several functions, such as the `featureRecognition`, `colorDetection` and `shapeDetection`, are associated with an identification parameter. The identification is generated from these functions in the robotic behavior layer. For example, as has been discussed the *global memory* stores the properties of features in the robotic behavior layer so as to recognize pictures using Harris-SIFT and RANSAC image processing techniques. We can use the identification of these pictures to match to the correct representation. However, some functions' identification parameter is relatively vague and not clear enough to represent a complete percept, and we need to combine this them in the interface layer. The XML schema in our case for this condition can be:

```
<descriptor sensor="Camera image" function="colorDetection">
    <para1="red"><interpreter>"red"</interpreter> </para1
    <para1="green"><interpreter>"green"</interpreter></para1
    ...
</descriptor>
<descriptor sensor="Camera image" function="shapeDetection">
    <para1="circle"><interpreter>"ball"</interpreter> </para1
    <para1="rectangle"><interpreter>"box"</interpreter></para1
    ...
</descriptor>
```

In the query and matching codes of the interface layer, the `interpreter` should be combined to form a complete percept, such as `redball`, `redbox`, `greenbox` and so forth.

The mechanism for knowledge acquisition and representation in the environment interface layer can support for a flexible, generic mapping approach. When a specific percept (e.g., `in(roomA)`) about the environment needs to be integrated in a task, we can just modify the XML database in the interface layer.

3.5 Information and Control Flow

A key issue in robot control architectures is the information and control flow. Each component in such an architecture needs to have access to the relevant information in order to function effectively. In line with the overall architecture of Figure 1 and layered architectures in general, different types of data are associated with each of the different layers. The information flow follows a strict bottom-up processing scheme whereas the control flow employs a strict top-down scheme. At the lowest level all raw sensory data is captured and then sent to the behavioral layer. The behavioral layer has a diverse set of functions to process the sensory data. By using the quantitative results of these functions, corresponding symbolic representations can be matched based on XML schema in the interface layer. Finally, the cognitive layer can use these symbolic percepts for deliberative reasoning.

Actions are selected for execution by the cognitive layer. These actions are translated by the interface layer into behaviors that can be executed by the behavioral layer which in turn are translated into motor control commands in the lowest layer. Action commands are symbolic messages. For example, the cognitive layer may decide to sent `goto(roomA)`. A message like this needs to be

parsed in the EIS interface layer to extract the parameters of the message. Subsequently, the behavioral layer will call the navigation component to actually perform the action. Because such behavior components take time to complete, moreover, it is important to monitor progress. Monitoring may happen at different layers. However, the behavioral layer typically plays a key role here. For example, due to the unreliability of walking (e.g. foot slippage), the path the robot follows needs to be continually re-evaluated and re-planned in real-time. The navigation component monitors and manages this action execution, until the robot arrives at the correct place of the room.

3.6 Synchronization of Perceptions and Actions

An inevitable problem in physical robot control is the synchronization problem of percepts and actions. In general, on the one hand, robots in physical world usually perceive its environment in real-time, and consequently sensors generate duplicated perceptual information that is useless for reasoning. On the other hand, decision making is usually faster than action execution on a robot. Typically, an action command has not been completed before another action command is sent for execution by the cognitive layer. Therefore, we need some means to synchronize the layers. As the functionality we need can be realized by the cognitive layer itself, we discuss some synchronization mechanisms that can be implemented in the cognitive layer.

The interface layer by itself does not guarantee that only those percepts needed by the cognitive layer are provided. In fact, this layer will typically produce a large number of (more or less) the same percepts repeatedly because the behavioral layer and the cognitive layer run in their own thread. For example, assuming the camera image frame rate is 30 fps, and the scene classification algorithm is fast enough, when the robot is standing in front of an open door and intends to figure out the state of the door. In this case, the cognitive layer will receive the same percept `doorOpen` 30 times in 1 second. Because many actions take typically much longer to execute on a robot, the repetition of this percept in such a short period of time does not provide much useful information to the cognitive layer. By sending these percepts nevertheless to the cognitive layer may still affect the reasoning and decision making in various ways. Apart from potential processing overhead, however, GOAL provides various ways to deal with repeated percepts. The following code illustrates how a programmer can handle a stream of percepts in a flexible way:

```
init module{
    knowledge{%can add other items in knowledge base that the robot knows.
        doorOpen:- not(doorClosed).
    }%can define init belief base, goals, and action specifications.
}
event module{
    program{
        if bel(percept(doorOpen), doorClosed) then insert(doorOpen).
        if bel(percept(doorClosed), doorOpen) then delete(doorOpen).
    }%can define the rules of putting percepts in the belief base.
}
```

The rules in the `event module` are also called *event* or *percept* rules. These rules specify how percepts can modify a robot's belief about its environment. In the code snippet above, once a percept `doorOpen` is received, the robot will hold the belief that this door is open. So, other duplicated `doorOpen` percepts coming from the EIS layer will not affect the reasoning and decision making.

Another aspect of the synchronization problem is the actions that are sent from the cognitive layer to the interface layer. Action executions usually have duration and take time to be accomplished. For example, if a robot's is to enters `roomA` and then show an arm raising gesture. When the robot holds a belief that it is `in(roomA)`, it should execute a command `gesture(arm-raising)`. However, showing an arm raising gesture cannot be accomplished immediately. An associated problem might be when the gesture has not been finished, another `gesture(arm-raising)` command will be generated for executing because the robot still holds the belief that when it is in `roomA`, it should perform this gesture. Furthermore, some actions can be run in parallel, but some actions cannot. This problem is very common during many action executions. To cope with it, the cognitive control layer provides a very flexible programming style for synchronizing actions:

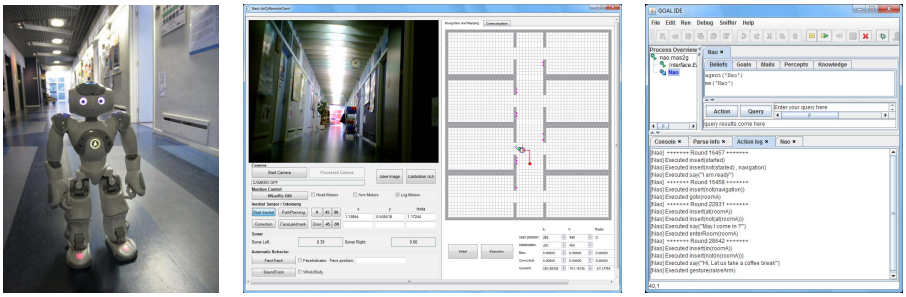
```
main module{
    program{
        if bel(in(roomA), static, not(showingGesture))
        then insert(showingGesture) + gesture(arm-raising) + say("Hi, Let's take a coffee break!").
        ... % other rules for reasoning and decision making.
}
```

Only does the robot believe that it is `in(roomA)`, `static` (i.e., not walking), and `not(showingGesture)`, the cognitive layer can generate the `gesture(arm-raising)` command. As the `say()` action can be performed in parallel with the `gesture()` action, such action can be generated so that they can be performed at the same time. We notice that the `showingGesture` will be inserted into the belief base of the robot once the robot begins to show the gesture. As a result, even if the `gesture(arm-raising)` has not been accomplished, and the robot is still `in(roomA)`, the cognitive layer will not generate a duplicated `gesture(arm-raising)` action command. By adding particular `beliefs` in the belief base, we can have a flexible approach achieve the synchronization of actions in the cognitive layer.

4 Navigation Task as an Example

To further illustrate our proposed architecture, we will use a navigation task as an example to explain how the agent-based cognitive control can be used to perform physical robots' tasks. The robot platform is a humanoid robot NAO (See Figure 3(a)) built by Aldebaran Robotics. In this task, the NAO robot acts as a message deliverer which is supposed to enter the destination room (e.g., `roomA`) and deliver a message to the people in the room.

The task is carried out in a domestic corridor environment. A predefined map has been built for the robot to localize itself. The robot begins walking from



(a) (b) (c)

Fig. 3. The navigation task: (a) humanoid robot NAO, (b) the GUI of the robotic behavior layer, (c) the cognitive reasoning in GOAL

a starting place: `placeB`, and the goal of the robot is to enter the destination `roomA`, to show an arm-raising gesture, and eventually to deliver a coffee-break message. Figure 3(b) shows the main GUI of the behavioral layer, and Figure 3(c) shows the GUI for the cognitive layer implemented in GOAL.

Figure 4 is the GOAL agent program for reasoning and decision making. The information that the robot knows about its environment consists of the `knowledge` and of the `beliefs`. The main differences between the knowledge base and the belief base is that the knowledge base is static and cannot be changed at runtime, whereas the belief base will be updated to keep track of the current state of the environment. Specifically, `knowledge` defines rules (e.g., `static:- not(walking)`), which cannot be modified; however, `beliefs` lists what the robot believes about the current environment (i.e., current belief is `in(placeB)`, but after a while the belief might be `in(placeC)`). The `goals` section lists the concrete goals that the robot has to achieve. Action specifications are enumerated in `actionspec` section. Each action specification also defines the preconditions `pre{}`: when the action can be performed, and the postconditions `post{}`: the effects of performing this action. In each reasoning cycle, the `event module` can modify the robot's belief base based on the percepts from its environment or other robots' messages. In the `main module`, action rules are defined as strategies or policies for action selection.

Although this navigation task is simple, it shows how the agent-based cognitive control uses the symbolic percepts, generated from uncertain, quantitative sensory data, to keep track of a robot's beliefs about its environment. Based on its beliefs, the robot can infer what actions should execute in order to achieve its goals.

5 Conclusion and Future Work

The navigation task that the robot has performed in the above section demonstrates the feasibility of using a cognitive layer to control physical robots by means of agent-oriented programming. It also demonstrates that the clean separation of sub-symbolic and symbolic layers via the interface layer. Although our

```
init module{
  knowledge{
    static:- not(walking).
    doorOpen:- not(doorClosed).
  }
  beliefs{
    in(placeB).
  }
  goals{
    messageDlivered.
  }
  actionspec{
    goto(Place){
        pre{static}  post{true}
    }
    enter(Room){
        pre{static}  post{true}
    }
    gesture(Behavior){
        pre{static}  post{true}
    }
    say(Text){
        pre{static}  post{true}
    }
    % ... add other action functions
  }
}
main module{
  program{
    if bel(in(placeB), static, not(inFrontOf(roomA))) then adopt(inFrontOf(roomA)).
    if a-goal(inFrontOf(Room)) then goTo(Room).

    if bel(inFrontOf(roomA), static, not(in(roomA)), doorOpen) then adopt(in(roomA)).
    if a-goal(in(Room)) then enter(Room).

    if bel(in(roomA), static, not(showingGesture))
        then insert(showingGesture) + gesture(arm-raising)+ say("Hi, Lets take a coffee break!").
  }
}
event module{
  program{
    if bel(percept(inFrontOf(Place))) then {
        if bel(not(inFrontOf(Place))) then insert(inFrontOf(Place)).
        if bel(inFrontOf(OldPlace)) then insert(not(inFrontOf(OldPlace)), inFrontOf(Place))).
    }
    if bel(percept(in(Room))) then {
        if bel(not(in(Room))) then insert(in(Room)).
        if bel(in(OldRoom)) then delete(in(OldRoom)) + insert(in(Room)).
    }
    if bel(percept(doorOpen), doorClosed) then insert(doorOpen).
    if bel(percept(doorClosed), doorOpen) then delete(doorOpen).

    if bel(percept(walking), not(walking), static) then insert(walking, not(static)).
    if bel(percept(static), walking, not(static)) then insert(static, not(walking)).

    % ... add other percepts
  }
}
```

Fig. 4. GOAL agent programming for cognitive control

general architecture is similar to the layered approaches that have been used in many robot projects [30–33], we believe that the use of EIS provides a more principled approach to manage the interaction between sub-symbolic and symbolic processors. Of course, it is important that the cognitive layer (agent program) needs adequate perceptions to make rational decisions given its specific environment. Our architecture provides sufficient support from various components dealing with perception, knowledge processing, and communication to ensure this. Especially, it provides a generic, flexible interface to map sensory data into symbolic knowledge for the cognitive layer.

Future work will concentrate on applying our proposed architecture for multi-robot teamwork in the Block World for Teams environment [34], in which each robot can exchange their perceptions and share their mental states in the cognitive layer so as to coordinate their actions.

References

1. Levesque, H., Lakemeyer, G.: Cognitive robotics. Handbook of Knowledge Representation, 869 (2008)
2. Beetz, M., Mosenlechner, L., Tenorth, M.: CRAM - A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments, pp. 1012–1017. IEEE (2010)
3. Kelley, T.D.: Developing a psychologically inspired cognitive architecture for robotic control: The symbolic and subsymbolic robotic intelligence control system. Advanced Robotic 3, 219–222 (2006)
4. Hanford, S.D., Janrathitikarn, O., Long, L.N.: Control of mobile robots using the soar cognitive architecture. Journal of Aerospace Computing Information and Communication 5, 1–47 (2009)
5. Hawes, N., Sloman, A., Wyatt, J., Zillich, M., Jacobsson, H., Kruijff, G., Brenner, M., Berginc, G., Skocaj, D.: Towards an Integrated Robot with Multiple Cognitive Functions, vol. 22, pp. 1548–1553. AAAI Press, MIT Press, Menlo Park, Cambridge (1999/2007)
6. Tenorth, M., Jain, D., Beetz, M.: Knowledge processing for cognitive robots. KI-Künstliche Intelligenz 24, 233–240 (2010)
7. Tenorth, M., Beetz, M.: Knowrob - knowledge processing for autonomous personal robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, pp. 4261–4266. IEEE (2009)
8. Hindriks, K.: Programming rational agents in goal. In: Multi-Agent Programming: Languages, Tools and Applications, pp. 119–157. Springer US (2009)
9. `http://ii.tudelft.nl/trac/goal` (2012)
10. Hindriks, K.V., van Riemsdijk, M.B., Behrens, T.M., Korstanje, R., Kraaijenbrink, N., Pasman, W., de Rijk, L.: Unreal goal agents. In: AGS 2010 (2010)
11. Behrens, T., Hindriks, K., Dix, J.: Towards an environment interface standard for agent platforms. Annals of Mathematics and Artificial Intelligence, 1–35 (2010)
12. Shanahan, M., Witkowski, M.: High-level robot control through logic. Event London, 104–121 (2000)
13. Duch, W., Oentaryo, R., Pasquier, M.: Cognitive architectures: Where do we go from here? In: Proceeding of the 2008 Conference on Artificial General Intelligence, pp. 122–136 (2008)
14. Soutchanski, M.: High-level Robot Programming and Program Execution (2003)

15. Coffey, S., Clark, K.: A Hybrid, Teleo-Reactive Architecture for Robot Control (2006)
16. Lemaignan, S., Ros, R., Mösenlechner, L., Alami, R., Beetz, M.: Oro, a knowledge management platform for cognitive architectures in robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2010)
17. Burghart, C., Mikut, R., Stiefelhagen, R., Asfour, T., Holzapfel, H., Steinhaus, P., Dillmann, R.: A cognitive architecture for a humanoid robot: a first approach. Architecture, 357–362 (2005)
18. Anderson, J.R., Lebiere, C.: The atomic components of thought, vol. 3. Erlbaum (1998)
19. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial Intelligence 33, 1–64 (1987)
20. Benjamin, P., Lyons, D., Lonsdale, D.: Designing a robot cognitive architecture with concurrency and active perception. In: Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics (2004)
21. Avery, E., Kelley, T., Davani, D.: Using cognitive architectures to improve robot control: Integrating production systems, semantic networks, and sub-symbolic processing. System 77 (1990)
22. Laird, J.E.: Toward cognitive robotics. In: Proceedings of SPIE, vol. 7332, pp. 73320Z–73320Z–11 (2009)
23. Bekey, G.A.: Autonomous Robots: From Biological Inspiration to Implementation and Control. The MIT Press (2005)
24. Baillie, J.C.: Urbi: Towards a universal robotic low-level programming language. In: 2005 IEEE RSJ International Conference on Intelligent Robots and Systems, pp. 820–825 (2005)
25. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
26. Azad, P., Asfour, T., Dillmann, R.: Combining Harris interest points and the SIFT descriptor for fast scale-invariant object recognition. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4275–4280. IEEE (2009)
27. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, 381–395 (1981)
28. Mika, S., Schaefer, C., Laskov, P., Tax, D., Müller, K.R.: Support vector machines, vol. 1, pp. 1–33. Springer (2004)
29. Kurfess, F.: Integrating symbol-oriented and subsymbolic reasoning methods into hybrid systems. In: From Synapses to Rules-Discovering Symbolic Rules from Neural Processed Data, pp. 275–292. Kluwer Academic Publishers (2002)
30. Qureshi, F., Terzopoulos, D., Gillett, R.: The cognitive controller: A hybrid, deliberative/reactive control architecture for autonomous robots. In: Orchard, B., Yang, C., Ali, M. (eds.) IEA/AIE 2004. LNCS (LNAI), vol. 3029, pp. 1102–1111. Springer, Heidelberg (2004)
31. Arkin, R.C.: Integrating behavioral, perceptual, and world knowledge in reactive navigation. Robotics and Autonomous Systems 6, 105–122 (1990)
32. Connell, J.H.: SSS: a hybrid architecture applied to robot navigation, vol. 3, pp. 2719–2724. IEEE Comput. Soc. Press (1992)
33. Gat, E.: Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots, pp. 809–815. Citeseer (1992)
34. Johnson, M., Jonker, C., van Riemsdijk, B., Feltovich, P.J., Bradshaw, J.M.: Joint activity testbed: Blocks world for teams (BW4T). In: Aldewereld, H., Dignum, V., Picard, G. (eds.) ESAW 2009. LNCS, vol. 5881, pp. 254–256. Springer, Heidelberg (2009)