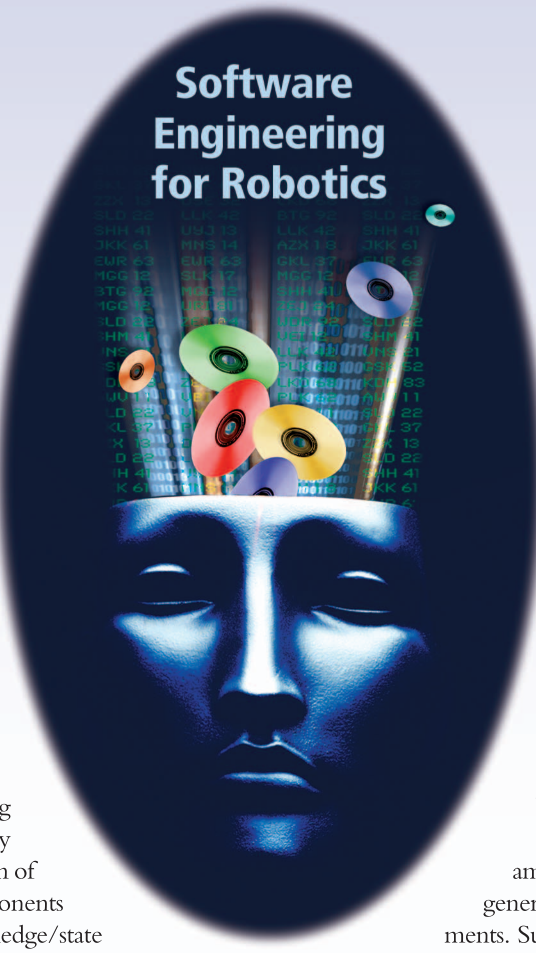
Cognitive Map Architecture

Facilitation of Human–Robot Interaction in Humanoid Robots



Among the challenges of building robots for everyday environments is the need to integrate diverse systems and subsystems. Here, we describe a step in this direction: the Cognitive Map robot architecture. It supports a flexible multicomponent system that can be dynamically reconfigured to handle different applications with minimal changes to the system. Runtime activation of traditional and hybrid robot architecture paradigms for any particular task is supported. Our method of isolating the communication interface within a single application programming interface (API) layer supports loose coupling between components, allowing easy integration of legacy code and expansion of existing systems. We classify the components into four main roles: perception, knowledge/state representation, decision-making, and expression. Interaction, Task Matrix and Multimodal Communication are modules built in this system for facilitating human–robot interaction with the humanoid robot ASIMO built by Honda Motor Co., Ltd. We describe the key ideas behind the architecture and illustrate how they are implemented in a memory card game application where people interact with ASIMO. Through our experience and comparison with alternative approaches, we show that the Cognitive Map architecture significantly facilitates implementation of human–robot interactive scenarios.

Digital Object Identifier 10.1109/MRA.2008.931634

The goal of developing flexible, versatile humanoid robots capable of coexistence with humans is a challenge, which nonetheless drives many roboticists to work with what typically is a highly temperamental combination of hardware and software. In addition to the many shared problems that the humanoid robots have with their nonhumanoid brethren, there are many challenges unique to humanoid robots. Chief among these is that they are intended for general task execution in everyday environments. Such robots must have a high number of degrees of freedom for flexible manipulation and navigation, a variety of sensors to gather information about their environments, and the ability to interact with people using natural modes of communication. These are important requirements that strongly dictate the design of the robot architecture.

How can we design an online reconfigurable software architecture capable of reusing components in different applications or interaction scenarios? One approach is to isolate

BY VICTOR NG-THOW-HING, KRISTINN R. THÓRISSON, RAVI KIRAN SARVADEVABHATLA, JOEL A. WORMER, AND THOR LIST

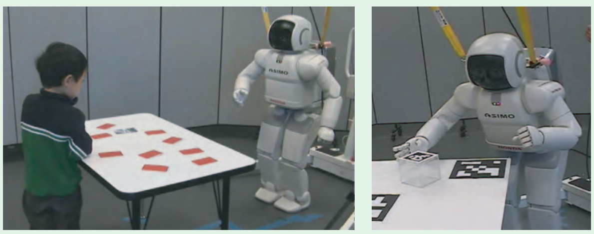
1070-9932/09/$25.00©2009 IEEE

application-specific details from reusable functionality, eliminating the need to rewrite entire parts of the system each time a new application is targeted. We have followed this methodology in building several applications, including an interactive memory card game [1] and a multimodal push planner for moving blocks around a table [2] (Figure 1).

Human–robot interaction research features the humanoid robot as an embodied intelligent agent that uses natural forms of Multimodal Communication with humans, such as delivering and understanding speech and gestures. This requires the need for parallel processing and time synchronization capabilities for analyzing and synthesizing multiple data streams. In addition to people, many tasks also require interaction with passive and active objects in a potentially unknown environment. The goal is not just to build a geometric representation of the environment for collision-free navigation, but also to detect, identify, and reconstruct objects for potential manipulation and reasoning with the environment. These problems require efficient designs to handle and share information throughout the system, as it flows from the sensors into various knowledge representation schemes to be acted upon by decision-making systems responsible for task-related motions on the robot.

Most of the technical requirements described earlier are still active research problems. Humanoid robots require an impressive integration of many disparate technologies that ultimately require them to work smoothly together to accomplish a task. No single research laboratory exists that can claim to be a master of all these research areas. Therefore, to pursue a closed, proprietary strategy for system development in this area would be both time-consuming and self-defeating. Yet researchers often invest considerable time and effort developing their own software prototypes, frameworks, and test-beds, including supplementary code libraries. They are comfortable and most efficient working in their favorite development environment (operating system, compilers, and build systems). The choice of environment can be driven by practical constraints such as driver availability for specific hardware or needing to use declarative languages versus procedural ones.

Many humanoid robots are comprised of several computers with different operating systems. Motor control is typically handled with a real-time embedded operating system. Camera sensors are driven with software on Linux or Windows operating systems. With current technology, it is difficult to build a single monolithic system with all tasks running on a single computer. Better load-balancing is achieved with distributed systems.



(a) (b)

Figure 1. Applications built with the Cognitive Map: (a) memory game and (b) multimodal push planner.

We think that any effort to attempt a standardization of robot components at all levels is misguided. In many areas, no consensus has yet emerged on best practices for solving problems such as localization, motion planning, or object classification. Any attempt to suggest researchers to rewrite their own software to a uniform standard will most likely be met with resistance due to the amount of time required to rewrite software perfectly adequate for their own current needs or that a considerable amount of time and money has already been invested in its development. Many people working on architectures for robots realize the importance of designing standardized robot components but have different convictions on what the form of that standard should be.

Acceptance of this fundamental situation played an important influence in the design and strategy of both the informational flow and structure of our robot architecture and the software engineering choices we made. To this end, we have developed the Cognitive Map robot architecture that minimizes the amount of rewriting of existing legacy software for integration. The Cognitive Map can be thought of as a centralized information space for connected components to contribute both internal and environmental state information. We leverage several successfully proven concepts such as blackboard architectures [3] and publish-subscribe based messaging [4] to develop a flexible robot architecture that exhibits fault-tolerance, easily substituted components, and provides support for different structural paradigms such as subsumption, sense-plan-act and three-tier architectures [5]. Our multicomponent distributed system has system components that are loosely coupled via message-passing and/or continuous data streams. This architecture was implemented on the humanoid robot ASIMO [6] manufactured by Honda Motor Co., Ltd.

We review various forms of communication middleware and component models in the next section. The "Architecture" section provides an overview of our architecture and considerations in its design. The "Scenario Design" section details the process from conceptualizing an interactive application to its instantiation in the robot architecture. The "Components" section singles out several important high-level components that play a significant role in many of our interactive scenarios. Finally, discussions and conclusions are presented.

Previous Work

Over the evolution of robot architectural design, one important structural theme has persisted: component models and their interconnectivity through distributed systems. Component models provide a software construct for encapsulation of elements of a robot's functionality or behavior and are represented with strictly defined interfaces where communication between components is done exclusively through message passing. As long as the interface is adhered to, the actual implementation details and environment can be opaque to the other components. Because of the sheer amount of parallel, collective computation required, components often exist on different processing, and storage elements. Many different robot architectures have been designed based on this basic idea but differ in the design of the component model. The design of the components' interface influences patterns of message interchange, representations of information, and the granularity of the components.

Component Model Frameworks

Player 2.0 [7] is the latest revision of a popular robotics development platform that provides standardized abstraction interfaces for robot sensors and actuators. The original Player [8] provided a single server for multiple clients of a robot's devices, whereas Player 2.0 allows multiple servers. However, servers are not allowed to have cyclic dependencies for information. In contrast, our architecture treats components as semi-independent agents, which can exchange information with each other via one or more blackboards.

Open architecture humanoid robotics platform (OpenHRP) [9] follows the common object request broker architecture (CORBA) model for interprocess communication between components. The structure of CORBA's interface definition language (IDL) provides an abstraction for programming language independence and distributed system communication. However, since the IDL promotes a remote procedure call protocol, the information and execution flow is restricted to client–server interactions. This makes component activity dependent on external components calling its functions, which in turn makes it difficult to design architectures with concurrent independent behaviors. Furthermore, the coupling between components is stronger than it needs to be as it is necessary for one component to know the available functions of another component to interact with it. In contrast, our Cognitive Map architecture follows a publish-subscribe protocol that allows looser coupling between components. OpenHRP also identifies several important components for humanoid robots, focusing more on motion generation: collision checking, dynamic simulation, motion planning, and controllers. The Cognitive Map has similar components, but many more are added for perception and decision-making, especially for targeting human–robot interaction.

The Microsoft Robotics Studio [10] treats components as services that can communicate asynchronously and run concurrently using the concurrency and coordination runtime (CCR) asynchronous programming library and decentralized software services (DSS) application model. Components must organize their state information and dependencies with other components using standardized elements such as service handlers for each type of incoming messages, partners for components it works with and service state for accessible component information. The advantages of this standardization of component parts are that components can have an easier time self-discovering the capabilities of other components. However, the burden is placed on developers to conform their existing legacy code into the structure dictated to by the component model. Furthermore, since Microsoft Robotics Studio is built upon managed runtime code libraries dependent on the Windows operating system, the flexibility of running components with other operating systems like Linux is limited. In contrast, the Cognitive Map uses an XML-based communication protocol and allows components to be implemented on a variety of programming languages (C, C++, Java, C#) and operating systems (Windows, Linux, MacOSX).

The brain bytes component model (BBCM) and brain bytes data model (BBDM) [11] were designed to encapsulate processing and data roles respectively for intelligent systems. During system design, the integrated system architecture is conceived first, followed by populating the design with modules that meet the functional requirements of the architecture. As BBCM components can be very simple in function, there can be hundreds of components in the system. Any existing software should be reimplemented to follow the component interface and functional constraints dictated by the system design. In contrast, our systems are built from storyboards that visualize the desired behavior of our robots in interactive scenarios. Key low-level component technologies are identified and created by modifying legacy code whenever possible. Several high-level components are then designed or reused that coordinate and collect the information produced by these lower-level components to produce new information. Consequently, our systems exhibit a smaller number of components with more encapsulated behavior in each component, simplifying the abstract view of the overall system design.

Human–robot interaction research features the humanoid robot as an embodied intelligent agent that uses natural forms of Multimodal Communication with humans, such as delivering and understanding speech and gestures.

Universal real-time behavior interface (URBI) [12] allows a robot to be represented as an URBI engine that can process execution scripts residing locally or sent to it via TCP/IP from remote clients. The components follow an object-oriented abstraction interface called *UObjects*, which nicely expose their functionality with object-oriented programming syntax features in the scripting languages of the engine. The URBI engine allows concurrent, event-driven execution of behaviors. Existing code must be converted into a UObject template. One key conceptual difference for the Cognitive Map is that it requires minimal changes to the legacy code. Rather than restructure existing legacy code to fit the component interface, the component interface for the Cognitive Map is accessed with a single API, whose routines are called within the legacy code.

Traditional Robot Architectures

Three dominant robot architectural paradigms are currently being used extensively. The sense-plan-act paradigm introduced in the Shakey robot [13] features three distinct stages of operation. This strict decomposition is not very suitable for dynamic environments. In response to this, the subsumption architecture [14] proposes building robot architectures from a collection of interconnected low-level behaviors, where sensor outputs are directly connected to actuators. Higher-level behaviors could then override or subsume the lower level behaviors. However, it is difficult to specify long-term behaviors or optimize plans consisting of multiple tasks with this kind of architecture. Layered architectures like 3T [15] attempt to combine the low-level behavior layers with high-level planning layers by introducing an intermediate executive layer for sequencing tasks. Some software frameworks for robots, like CARMEN's support of 3T [16], directly adopt a specific architectural paradigm to follow.

We have found that a single architectural design does not efficiently implement all tasks equally well. The $S^{\star}$ approach to behavior-based control argues that access to high-level task-based knowledge for perception components is important especially for implementing attention mechanisms [17]. Although this approach could not be directly supported in idealized implementations of the architectural paradigms described earlier, the ability of components in the Cognitive Map to subscribe and publish to any other component in the system permits $S^{\star}$ and other alternative controller arrangements to be implemented. Our communication subsystem is designed to allow dynamic reconfiguration of our components so that any one of these paradigms can be activated for a particular task to be done. This approach derives partly from the Ymir agent architecture for multimodal dialog and interaction [18], which proposed groups of preceptors, deciders, and actions/planning components operating and interacting in parallel, as well as transversal priority layers that cut across perception, decision, planning, and action. Like in Ymir, Cognitive Map components can potentially accept messages from any other component, removing the need to partition components into canonical software layers.

Architecture

To manage shared information, the Cognitive Map architecture is built on the Psyclone Whiteboard system [19], which combines the shared information concepts of a blackboard architecture [3] with data streams that can be shared, have their data samples time-stamped for synchronization, and data content transformed (e.g. coordinate conversion) or selectively screened while being transmitted between components. One or more blackboards can be located on the centralized server. In our architecture, we use two blackboards, CognitiveMapWB and TaskWB, for handling environment state and transient command messages, respectively (Figure 2). Different blackboards can also be assigned to each level in a layered architecture to explicitly partition shared information based on its functional level of operation.

Messages can be sent individually or placed in continuous data streams, which feature fixed semantics and less processing overhead per message. Components can publish and subscribe to messages and streams on the blackboards. An important point is that publishers and subscribers do not have to know about each other, making this form of coupling looser than CORBA-based or DSS-based (see the "Previous Work" section) component models. The advantage of this loose coupling is that components can be restarted (or even substituted with other components sharing the same interface) without affecting the rest of the system. Our architecture follows the constructionist design methodology (CDM) [20]. CDM was developed to help in the creation of systems capable of a large number of functionalities that must be carefully coordinated to achieve coherent system behavior. CDM is based on iterative system construction where components are incrementally added to a network of named interacting modules.

To minimize the effort of integrating existing legacy code when converting them to components, we developed a single library called the CogMapApi that manages message and stream communication within the Cognitive Map. To handle incoming and outgoing message traffic, a component typically calls one or both of two functions, UpdateFromCogmap and UpdateToCogmap, that are inserted into the beginning and end of the original main processing loop of the component, respectively (Figure 3). If a component exclusively publishes or subscribes to messages, only one of these is necessary. In our experience, a component can be integrated into our system

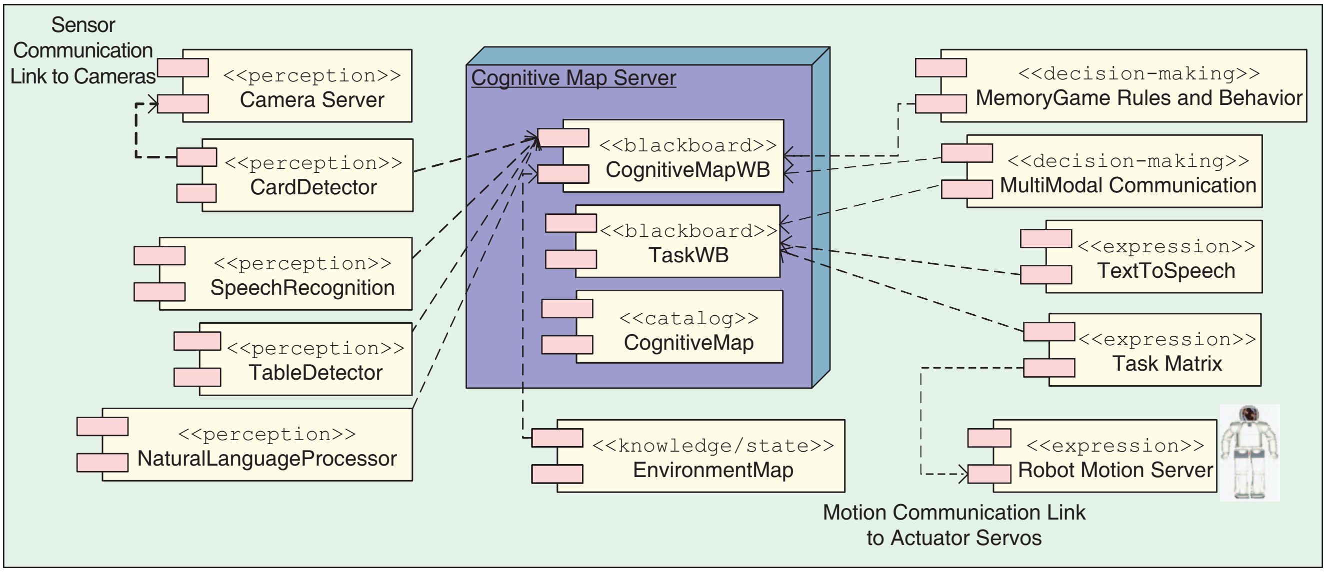


Figure 2. *Overview of the Cognitive Map robot architecture: Representative components depicted showing connectivity to the central Cognitive Map server. Component types are labeled based on name. Arrow directions depict dependencies on various interfaces (not message directions).*

within a few days. This is possible because the adopted strategy leaves the majority of existing legacy code unchanged by isolating Cognitive Map-related communication to CogMapApi calls in the two functions described earlier.

The Cognitive Map does not directly support real-time communication (with guaranteed response times to events), although it does support time-aware communication. Our architecture uses specialized communication libraries in situations where more direct peer-to-peer communication is required and were it is not necessary to store information in a centrally accessible blackboard. Typically this includes communications for sensors and actuators, such as streaming camera video directly to components or providing a low-level motion interface for sending joint angle trajectories to the robot. In the case of streaming video from cameras, components handling different vision tasks can subscribe to the same video stream from a camera server onboard the robot. They can then reside on dedicated computers for faster distributed computation while reporting their computed results back to the Cognitive Map blackboard. In cases where hard real-time is needed, components are implemented together on a real-time operating system with a dedicated communication link to the rest of the Cognitive Map.

Messages

The Cognitive Map features a centralized server to handle high-level message dispatching and registration of connected components. Components can either reside externally from the Cognitive Map, using TCP/IP socket-based communication, or be dynamically loaded internal components, communicating directly through memory. Following the OpenAIR specification [19], messages have a type that provides the selection criteria for subscribing. In order for components to be aware of the available messages in the system, an ontology of all the messages needs to be maintained. Messages types are hierarchical with each level delimited by a period (.). For example, for table events, we have `percept.vision.physobject.table.appear` and `percept.vision.physobject.table.touch` events.

The contents of the message can consist of various primitive and aggregate data types (real, string, integers, tables), but can also consist of objects with attributes. In particular, the information about both tangible and intangible objects is encapsulated in a CMObject data object (Figure 4). CMObjects can represent physical objects identified from sensory input or conceptual objects generated by the algorithms of a component (e.g., observed actions). Physical objects can have 3-D pose and geometric information and they can be symbolically labeled with an object type if it can be identified. Objects of a specific type can also have custom fields associated with them. For example, a table object has its length and width parameters for the tabletop to allow reconstruction of its geometry from a relatively small set of parameters. Messages can correspond to pure information about the sensed world (e.g., object poses), commands to specific components (e.g., task execution commands) or event notifications (e.g., person touching a table).

In addition to its basic dispatching role, the Cognitive Map provides three mechanisms for processing messages: indexers, deciders, and detectors. Indexers provide different ways to access stored data coming into the Cognitive Map via streams. By default, samples on a stream are accessed by timestamps and frame count. Indexers can provide other search criteria for accessing an object with high-performance indexing, such as organizing objects based on their coordinate positions. Detectors are instantiated dynamically by the Cognitive Map upon requests from other system components, to set up tests for world conditions and events based on data from one or more Indexers. Typically, the creation of a detector automatically initiates the creation of an indexer.

A detector implements a Boolean function that can produce answers to specific questions about the data in a stream. One indexer is created to facilitate this process upon the creation of a detector. The criteria for a detector can originate from a component and be dynamically specified at runtime. In our system, detectors can be specified using preset Boolean operators written in C or be entirely scripted and interpreted at runtime using the Lua language [21]. All samples coming through a stream monitored by the detector are evaluated and allowed to pass if the Boolean expression has a true value. This mechanism can be used to reduce message traffic. For example, a component responsible for manipulating objects on a table can use detectors to report new objects exclusively within the bounds of the table top. Deciders subscribe to information from one or more detectors, or other deciders, and make decisions on how to respond to events. They can then report their decisions back to the Cognitive Map. In the "Interaction" section, we describe how these three mechanisms can be used for a specific application like the memory game.

Scenario Design

The architecture describes the components and communication middleware between these components. However, the

```
void MyComponent::MainLoop() // Where all the work
                             // gets done
{
 UpdateFromCogmap(); // Poll messages/streams from
                     // Cognitive Map
 DoProcessing(); // Examples: feature detection,
                 // decision-making, planning, etc.
 UpdateToCogmap(); // Publish information to Cognitive Map
}
```

Figure 3. *Main loop of component for processing incoming (UpdateFromCogmap) and outgoing (UpdateToCogmap) messages from the Cognitive Map.*

id	Victor-Mug	
type	cup	
pos	(3.5,3.2,7.0)	
orientation	(0,0,0,1)	
training	view1	Victor-Mug-001.png
	view2	Victor-Mug-002.png

Figure 4. *Sample table representation of the message contents stored in the CMObject corresponding to a cup object with multiple viewpoint images.*

physical instantiation of the architecture—the specific components and message passing behavior during robot execution depends on the nature of the active robot application. As a robot switches application modes, the Cognitive Map reconfigures itself by changing the set of components it interacts with. In the following sections, we will ground the discussion of our architecture by reference to an interactive memory card game that was developed between ASIMO and a human player [1].

Memory Game

The Memory Game (Figure 5) features a deck of matching pairs of cards that are shuffled and placed face down on a table. Players take alternate turns picking two cards in an attempt to find and collect matching pairs. A player who successfully finds a match keeps the cards and is entitled to another turn. If the player does not find a match, the other player starts his or her turn. When one player achieves a majority of the cards or both players tie, the game is over.

Design

Our design process begins with the construction of a branching storyboard (Figure 6) that depicts the desired observed behavior of both the robot and humans involved in the scenario. Every attempt is made to predict different conditional behaviors and responses to those situations. For example, the storyboard shows how the robot would behave if it won, what happens if it failed to see a detected card or how to respond if the human player asks for help. This process allows us to identify what technical components are required and what information they should publish or subscribe to for accomplishing their role within the system.

Relating to the Cognitive Map, the storyboard aids both in the design of individual components and their message interfaces between each other. For example, if dialog is present in the storyboard, this implies the need for both speech recognition and speech synthesis components. The storyboard in Figure 6 further suggests that a simple input text message interface to the speech synthesis component would suffice for the dialog requirements of the scenario, preventing the implementation of unnecessary features or an overly complex interface to the component. The types of gestures that we wish the robot to perform also have important implications for motor control and sensing. Pointing gestures require both awareness of object position in the environment and an end-effector based control scheme for positioning and orienting the robot's hand.

In the design of interaction components, which will be further described in the "Interaction" section, the storyboard identifies all individual states that can occur during the scenario as well as the events that can trigger transitions between these states. This suggests what would be the most appropriate decision-making scheme to adopt within the interaction component. In the memory game, a finite state machine could be used to keep track of the current contextual game state and expected transition events

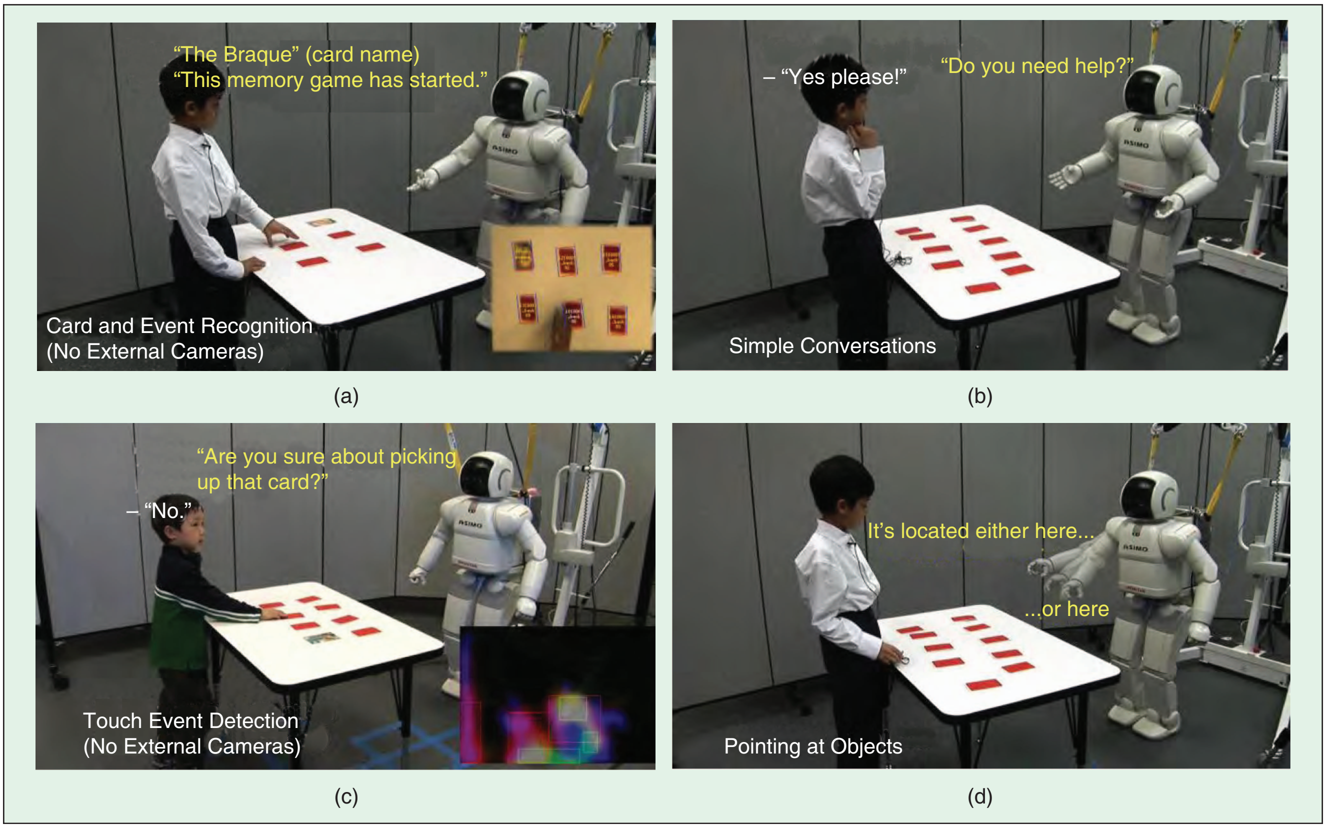


Figure 5. *The memory game application features integration of many components to create a human–robot interactive experience: (a) activity detection (recognition of flipped cards), (b) Multimodal Communication with speech and gestures, (c) proactive behavior (warning about touched cards) and (d) spatial tasks using information in the environment map (pointing at cards). Yellow and white text denotes dialog spoken by ASIMO and the human player, respectively.*

that are likely to occur from each state. The transition events between states in the storyboard identify what types of messages need to be generated by the components of our Cognitive Map architecture. In Figure 6, a transition occurs when a player picks a card, implying that a perception module for detecting cards would need to detect physical card motion or flipping and publish these message events to the Cognitive Map. If the storyboard described a simpler application that does not need to keep track of history, a basic action-selection table could be implemented as the interaction component instead.

Components

Components in our Cognitive Map architecture tend to feature a combination of four broad roles: perception, knowledge/state representation, decision-making, and expression (see Figure 2 for examples). Perception components include low-level sensor outputs and various feature extractors that extract higher-level information from the sensor data. For example, a component that performs face detection and tracking, or that returns object identification, would be in this category. Knowledge/state representation components use features to assemble higher-level information such as internal or external environmental state information. This information can also be transient, such as a list of active tasks the robot is performing, or it may be long-term information, such as a database of recognized objects encountered during the robot's operational lifetime. The Cognitive Map supports generic database objects called catalogs (Figure 2) for accessing persistent information. The decision-making components make use of the stored information or events from the perception components to decide what actions to perform in the form of new motor and nonmotor tasks. Finally, expression components result in physical observed behavior from the robot, including motor task execution and speech utterances. Components can have fairly specialized behaviors such as text-to-speech conversion or an object detector. However, in our experience, we have identified several high-level components that have general use across several applications. We describe these in the following subsections.

Debugging distributed systems is challenging because of the difficulty in isolating the source of observed incorrect robot behavior.

Interaction

For interactive applications, the scheduling of tasks must be dependent on events in the robot's environment. It must be assumed that any task can be interrupted by sudden changes in human behavior, such as a new verbal request, while the robot is in the middle of executing a task. Humanoid robots must have the capability of switching between different interactive tasks. This means that the sequencing of tasks for a specific

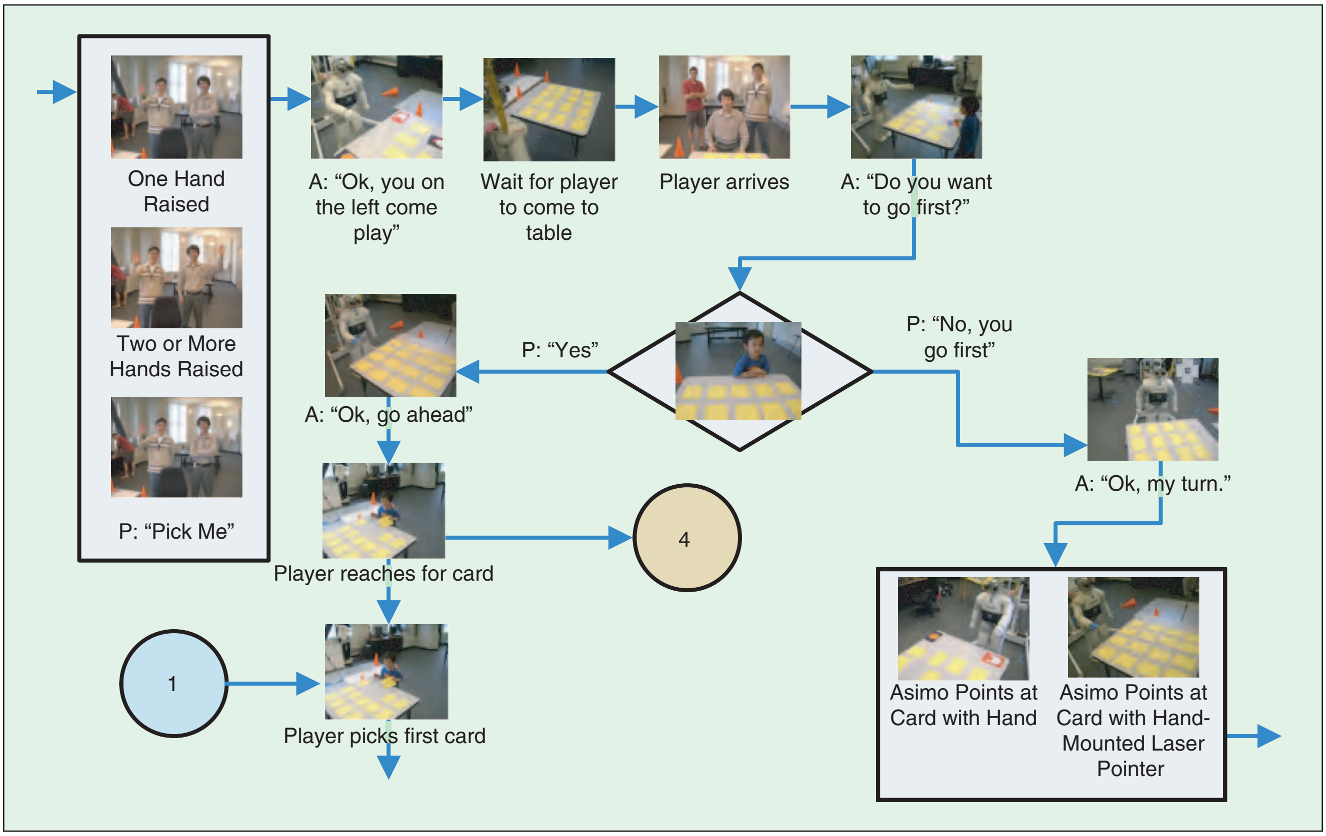


Figure 6. *Portion of the storyboard used to design the memory game. The circled 1 and 4 are references to different scenarios not shown here.*

Messages can be sent individually or placed in continuous data streams, which feature fixed semantics and less processing overhead per message.

interactive application must be modularized to allow interchange of robot behavior.

The Cognitive Map allows interaction components to be created to orchestrate different forms of interaction from turn-based games, query-response interchanges, to script-based scenarios with conditional branching based on how a human responds. The interaction component is a decision-making component and is usually the main coordinating component that orchestrates behavior in response to events under different contextual situations. To aid in the decision process, this kind of component subscribes to inputs from perception components and sends queries to the knowledge and state representation components. Once an action has been decided, the interaction component sends appropriate command messages to the expression components.

Consequently interaction components often play the role as deciders, as described in the "Messages" section. In the memory game, the interaction component subscribes to different detectors such as the table for recognizing the necessary conditions to start a game or a card detector for notification of card appearance, removal, and flipping events. If we place other noncard objects on the table, we can use the detector mechanism to filter out these objects so that the detector stream will only report card-related object events. If the player chooses to discard cards in a specific region of the table, an indexer can be created that organizes the cards by coordinate positions so that a detector can be established that filters out card events occurring in the discard region of the table. These mechanisms allow the interaction component to contain simpler logic by eliminating the need for extensive special-case handling of nonrelevant events.

The structure and implementation of the interaction component depends on the nature of interaction required by the application. For the memory game, we initially used a single finite state machine (FSM) to represent the entire game state (Figure 7), modeling both the player and the robot's state as well as several special-case scenarios, such as the player asking for help or ASIMO warning the player they are about to pick a bad card. We found that as we increased the complexity of the interaction, the number of states and transitions in our FSMs increased dramatically and had negative implications for scalability.

We were able to refactor the interaction component to keep multiple state machines for different entities in the game (Figure 7), featuring one state machine for the state of cards on the table and one for the rules of the game. This approach simplified the structure of the game, and increased robustness by allowing the card table state machine to focus on valid card layouts while the game rules state machine separately monitored whether it was currently the robot or the player's turn.

Environment Map

The environment map is a knowledge/state representation component (Figure 2) that collects pose information from objects in the scene, such as the table and its cards. It unifies the different position information for all objects in the scene into a common reference frame, allowing important spatial operations to be performed such as deictic gestures and collision avoidance.

Task Matrix

The Task Matrix (Figure 2) is an expression component that serves to map high-level symbolic commands to the low-level motor commands that physically realize the task. It consists of a collection of parametrized tasks ranging in complexity from simple following of joint angle trajectories to manipulation tasks complete with motion planning to find collision-free paths in the workspace. The Task Matrix separates out reusable task programs from the application specific interaction components. The ability to specify parameters for a task allow the task's generic definition to be applicable to a particular environmental situation. For example, the generic pointing task can be made to point at different objects in the scene by specifying the target object as a parameter. The Task Matrix queries the environment map to resolve symbolic parameters like a card's name into its 3-D pose in the environment.

Task programs within the Task Matrix are implemented as dynamically loaded plug-ins that are loaded on demand as a task execution request is made. Rather than enforcing a single control paradigm for all tasks, the Task Matrix allows tailored controllers for different tasks, which is important for humanoid robots. For example, a walking task can use a more simplified 2-D planar planner for simple navigation whereas a complex pushing task required 3-D obstacle information so that the upper body can avoid self-collision or colliding with a table.

Simplified resource conflict resolution is provided by making sure the Task Matrix's different tasks do not compete for the same kinematic chains. Other runtime checking for tasks can be done such as verifying that an object to be manipulated is present in the stored environment map of the robot. To avoid code duplication, common libraries for motion control such as motion planners or inverse kinematic routines can be shared between all tasks. In the memory game, the Task Matrix is used to store a library of gestures, both representing free motions and spatial tasks that respond to changes in the environment.

Multimodal Communication

Specific to humanoid robots is the need for natural, multimodal forms of interaction. Humans typically combine different modalities for effective communication. For example, we commonly gesture with our hands while engaging in conversation. The Multimodal Communication (MMC) component coordinates speech and gesture expression. Interaction components send parameterized utterance types to the MMC component, which internally converts these requests to spoken text and/or gesture messages that are forwarded to text-to-speech and the Task Matrix components respectively. The choice of wording is influenced by a combination of style directives from the interaction component and internal state information in the knowledge representation components.

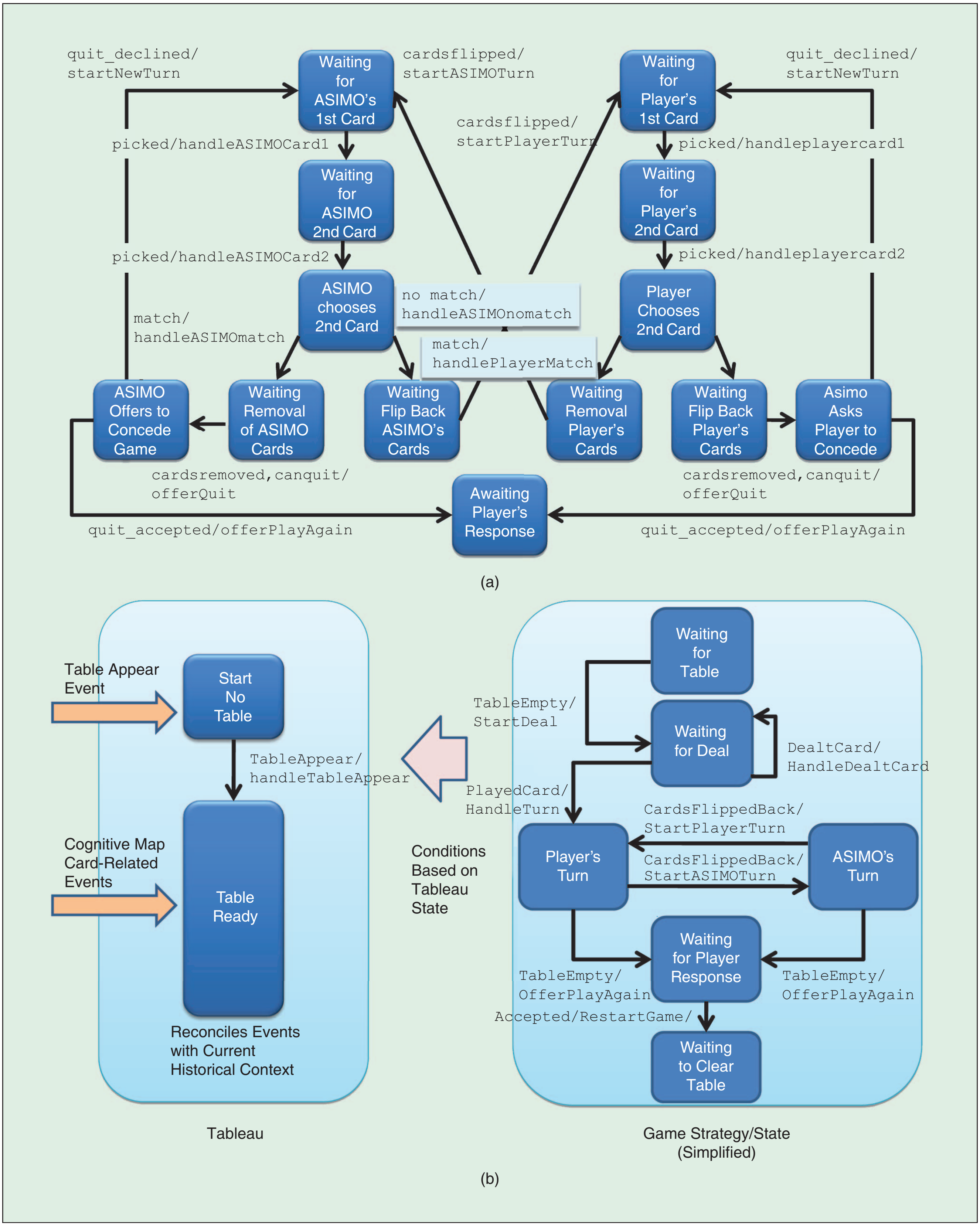


Figure 7. *Evolution of the interaction component's finite state machines. Transitions denote condition/action pairs. (a) Original finite state machine. (b) Refactored finite state machine.*

The MMC is a decision-making component that accepts utterance-type messages from the interactive components, specifying the content of the message to be spoken. For instance, the directive Indicate(<objects=card10>,<style= urgent>), is processed by the multimodal module to generate the spoken words, "I choose this card," and the concurrent deictic gesture of pointing at the physical card on the table. Intonation changes can be placed on the word "this." The developer responsible for the interactive module does not need to worry about the mechanism for recovering spatial position of the card nor the choice of words spoken.

Multimodal Communication can be used to remove characteristic repetitive behaviors from the scenario: By forcing the developers of the interaction module to think only about the content of the communication and deferring the style to the MMC, the robot can behave in slightly different ways under the same game state conditions, greatly increasing the flexibility of the system. Since the interactive module does have knowledge of contextual state information, the style parameter can be used to offer hints to the Multimodal Communication on how to express the message. For example, if <style=urgent>, the robot can choose to modify the text-to-speech parameters to adjust the speed of spoken text. Localization and culture-specific interaction issues can also be handled by the MMC. Since the multimodal module can subscribe to messages that indicate information about the player (name, age, gender, nationality), this information can be used to select appropriate gestures such as bowing to the Japanese and holding out a handshake to North Americans and Europeans. Personalization in spoken words can also be handled, such as "John, do you need help?" and different phrasing can be chosen for requests directed at children versus adults.

In contrast to the application-specific interaction component, the multimodal component provides generic response mechanisms that can be reused across unrelated scenarios. For example, it provides the generic response mechanism Indicate. This mechanism can handle different kinds of objects the robot can expect to encounter (cards, fruits, household objects) and their location in the environment, number of entities (singular, multiple) and semantics of indication (refer to each entity or collectively to all of them). Such a mechanism naturally leads to a parsimonious framework without sacrificing the robot's expressiveness.

Discussion

The design of the Cognitive Map robot architecture can significantly facilitate the implementation of human–robot interactive scenarios. We will now reflect on our robot architecture from two perspectives: component design and communication of information between components.

Reusability

A general software engineering principle we adopted was removing application-specific details from as many components as possible. In cases where application-specific details are unavoidable, such as knowledge of the rules of a game, they were isolated to a single module, allowing all changes to the application to be made in one location. This decision allowed us to reuse many components for other scenarios.

In general, components that have communication functionality are often reused because of their general importance in human–robot interaction. The text-to-speech and Multimodal Communication components were reused in a conversational application where ASIMO answers questions about the research ongoing in our lab and its own technology. The Task Matrix and Environment Map were used together with an object recognition system to point at and identify objects on a table. We have also used the Environment Map for planning pushing motions to manipulate objects on a table [2]. The table detector component from the memory game was used in the pushing application (Figure 1) to update the Environment Map to allow the robot to navigate around the table while walking. The Environment Map provided up-to-date configurations of objects on the table, allowing the motion planner to re-plan when it was notified of changes in the table environment.

Abstraction

Certain components serve as intermediaries between high-level decision components and low-level robot behavior. Specifically, they transform high-level directives to low-level expressions of behavior. The Task Matrix allows high-level symbolic commands to be transformed into physically realizable actions. The Multimodal Communication component takes symbolic utterances and coordinates both speech and gestures. Superficially, this role seems similar to the executive layer in layered robot architectures like 3T. However, in our architecture the entire layer has been encapsulated and partitioned into several different components with clear responsibilities. This modularity allows these behaviors to be managed and maintained separately without refactoring other parts of the architecture. For example in the Task Matrix, by employing a plug-in based mechanism for expanding the number of tasks, and providing access to all tasks through a single component interface, a cleaner mechanism for dynamically adding and removing tasks is achieved. If the robot's hardware or joint configuration is modified, changes only need be made in the Task Matrix while keeping the rest of the system untouched. The Multimodal Communication module separates the content in the application from the style in which that content is expressed during communication. Any changes done in this component will result in immediately changed behavior in all applications that use it.

Information Flow

In the area of information communication between components, we have found that one-to-many patterns, where information published by one component is consumed by multiple components, offer a rich and powerful way to initiate complex, concurrent behavior in the robot. For example, during the memory game, the player may flip a card, generating a detected perceptual event which gets published by the card detector component to the rest of the Cognitive Map. Simultaneously, this information is handled by different component agents for their own purposes. The interaction module which keeps track of the game's states uses this information to determine if the robot should initiate its turn or wait for a second card to be drawn from the player. Meanwhile, the environment map uses the card flip information to update its knowledge of the card's identity and current position on the table.

Systems-Based Approach

The benefits of taking a systems-based approach to building an application enables new valuable information to be generated and shared as a result of the integration of several components. For example, in the memory game, a table detector publishes its 3-D pose information with respect to the camera as well as the transformed homographic image of the table top. The card-detector uses this image stream to report position of cards in terms of 2-D image coordinates. However, since the environment map module has access to both the table's 3-D pose information, the 3-D transformation of the camera with respect to the robot, and the 2-D positions of the cards, it is able to assemble this information to report back 3-D positions of cards with respect to the robot's reference frame, allowing the robot to point at the cards to indicate its intentions.

Independent Behavior

The amount of behavioral independence in each component agent seems to promote flexibility and robustness in the system. For example, a common problem with interactive tasks involving computer vision is determining when to disable various specialized object detectors during the course of interaction. For example, if ASIMO moves his head to speak to the player, a table detector may incorrectly assume a table has been removed from the scene since it would lose track of the object. One solution is to allow the Task Matrix to report head motions to the Cognitive Map and allow individual components to decide how to appropriately respond to that information. In the case of the table detector, if it realized that the camera motion would be disruptive, it could choose to disable its processing until it was notified that head motion has stopped. On the other hand, another object detector component could decide to continue to operate if it has more robust tracking algorithms. Deferring many of the operational decisions to individual perceptual components simplifies the logic of higher-level components that interact with them.

Synergies

In the course of the memory game, we were able to easily substitute the card detector and game behavior components with faster and more robust implementations, without requiring changes in the rest of the system. This was possible with loose coupling. However, beneficial effects of the improved behavior do tend to affect overall system performance. A faster module that publishes messages consumed by many other components tends to improve overall system responsiveness since multiple lags due to processing delay are reduced. To improve overall robustness in an application, both component-level robustness and exploiting multiple sources of information are needed. For example, many vision components suffer from false positive detection events. However, by analyzing concurrent events and other surrounding state information, it is possible to identify and avoid such false positive events. The environment map can rule out objects that are physically implausible, such as a table that appears to be floating an unreasonable distance above the floor. A false card flip event in the memory game could be detected by checking if there is a coincident table touch event by the table detector.

Conclusion

The relative naturalness and speed of implementing various phenomena with the Cognitive Map architecture reinforces our confidence of its suitability for modeling human–robot interactive applications. This is achieved using a design that features components that exhibit behavioral independence and have abstract interfaces that permit the substitution and reuse of components. The publish-subscribe communication scheme facilitates concurrent and coordinated behavior in our robot.

The robotics research community is diverse and highly specialized. This has resulted in a focus on solving problems under a highly qualified set of conditions. With the Cognitive Map, we allow components to share information to aid in their individual processing. Introducing external sources of information to the system is sometimes seen as cheating or reducing the purity of the problem. In contrast, we believe that achieving higher levels of robust performance for interactive applications can only be done using a systems-based approach where multiple sources of information can be combined to create new knowledge and confidence in the robot's understanding of its situation.

Debugging distributed systems is challenging because of the difficulty in isolating the source of observed incorrect robot behavior. In future work, we intend to develop a monitoring tool that will act as an additional but independent component in the Cognitive Map architecture. This component will allow visual inspection of the interrelationships between components at runtime. Allowing an operator to visualize the dependencies and flow of information can reveal the component that was the original source of incorrect information, instead of mistakenly attributing the problem to an intermediate component. Because our architecture easily combines perceptual elements, motor task generation and knowledge representation, we are using this framework in the investigation of task learning from observation—in fact, the Cognitive Map architecture was partially designed with these research problems in mind. By modularizing the behavior and structure of interactivity in the particular manner described here, we can more easily experiment with various mechanisms for interaction. By combining different interaction models, humanoid robots can begin to exhibit autonomous and adaptive behavior in their interactions with humans.

Keywords

Robot architecture, humanoid robots, communication middleware, Cognitive Map.

References

[1] V. Ng-Thow-Hing, J. Lim, J. Wormer, S. Ravi Kiran, C. Rocha, K. Fujimura, and Y. Sakagami, "The memory game: Creating a human-robot interactive scenario for ASIMO," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS '08)*, 2008, pp. 779–786.

[2] K. Hauser, V. Ng-Thow-Hing, and H. Gonzalez-Banos, "Multi-modal motion planning for a humanoid manipulation task," in *Int. Symp. Robotics Research (ISRR 2007), Preprints*, 2007, pp. 351–361.

[3] B. Hayes-Roth, “A blackboard architecture for control,” *Artif. Intell.*, vol. 26, pp. 251–321, 1985.
[4] A. Farinelli, G. Grisetti, and L. Iocchi, “Design and implementation of modular software for programming mobile robots,” *Int. J. Adv. Robot. Syst.*, vol. 3, no. 1, pp. 37–42, 2006.
[5] D. Kortenkamp and R. Simmons, “Robotic systems architectures and programming,” in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. New York: Springer-Verlag, 2008, pp. 187–206.
[6] Honda Motor Co.(2000). *Asimo year 2000 model* [Online]. Available: http://world.honda.com/ASIMO/technoloogy/spec.html
[7] T. H. Collett, B. MacDonald, and B. P. Gerkey, “Player 2.0: Toward a practical robot programming framework,” in *Proc. Australasian Conf. Robotics and Automation (ACRA)*, 2005. [CD-ROM].
[8] B. P. Gerkey, R. T. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” in *Proc. Int. Conf. Advanced Robotics (ICAR)*, 2003, pp. 317–323.
[9] F. Kanehiro, H. Hirukawa, and S. Kajita, “Openhrp: Open architecture humanoid robotics platform,” *Int. J. Robot. Res.*, vol. 23, no. 2, pp. 155–165, 2004.
[10] Microsoft Corporation. (2006). *Microsoft robotics studio*. Available: http://msdn.microsoft.com/en-us/robotics/default.aspx
[11] A. Ceravola, M. Stein, and C. Goerick, “Researching and developing a real-time infrastructure for intelligent systems—Evolution of an integrated approach,” *Robot. Autonom. Syst.*, vol. 56, no. 1, pp. 14–28, 2008.
[12] J. Baillie, “Universal programming interfaces for robotic devices,” in *Proc. Joint Conf. Smart Objects and Ambient Intelligence*, 2005, pp. 75–80.
[13] N. Nilsson, “Shakey the robot,” SRI, Menlo Park, CA, Tech. Rep. 323, 1984.
[14] R. Brooks, “A robust layered control system for a mobile robot,” *IEEE J. Robot. Automat.*, vol. RA-2, pp. 14–23, Apr. 1986.
[15] E. Gat, “On three-layer architectures,” in *Artificial Intelligence and Mobile Robots*, R. P. Bonnasso and R. Murphy, Eds. Cambridge, MA: AAAI Press, 1998.
[16] M. Montemerlo, N. Roy, and S. Thrun, “Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit,” in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS ’03)*, 2003, vol. 3, pp. 2436–2441.
[17] A. Rotenstein, A. Rothenstein, M. Robinson, and J. Tsotsos, “Robot middleware must support task-directed perception,” in *Proc. ICRA 2nd Int. Workshop on Software Development and Integration into Robotics*, Rome, Italy, Apr. 14, 2007. [CD-ROM].
[18] K. R. Thórisson, “A mind model for multimodal communicative creatures and humanoids,” *Int. J. Appl. Artif. Intell.*, vol. 13, no. 4–5, pp. 449–486, 1999.
[19] K. R. Thórisson, T. List, C. Pennock, and J. DiPirro, “Whiteboards: Scheduling blackboards for semantic routing of messages and streams,” in *Proc. AAAI-05 Workshop on Modular Construction of Human-Like Intelligence*, 2005, pp. 8–15.
[20] K. R. Thórisson, H. Benko, A. Arnold, D. Abramov, and A. Vaseekaran, “A Constructionist design methodology for interactive intelligences,” *AI Mag.*, vol. 25, no. 4, pp. 70–90, 2004.
[21] *Programming in Lua*, Lua.org, 2006.

Victor Ng-Thow-Hing received the M.Sc. and Ph.D. degrees in computer science in 1994 and 2001, respectively, from the University of Toronto, where he was a member of the Dynamics Graphics Project Lab. He is currently a research scientist at the Honda Research Institute USA in Mountain View, California. Previous research includes work in 3-D muscle modeling, digital human modeling, and computational paleontology. Victor’s current research focuses on developing autonomous high-level tasks for Honda’s ASIMO humanoid robot, Multimodal Communication interfaces and designing software architectures for robots specializing in human–robot interaction.

Kristinn R. Thórisson holds a Ph.D. degree in media arts and sciences from the MIT Media Laboratory. He is founding director of the Center for Analysis and Design of Intelligent Agents (http://cadia.ru.is) and associate professor at Reykjavík University, School of Computer Science. He has been researching artificial intelligence in academia and industry for two decades. His Ymir/Gandalf system, developed at Massachusetts Institute of Technology (MIT), presented pioneering ideas in the area of communicative architectures for integrated multimodal perception, knowledge, and action control. He has worked at LEGO, British Telecom and Columbia University, as well as several startups in New York, and consulted on robot technology for NASA. He is cofounder of Radar Networks Inc. and coordinator of the multiyear HUMANOBS research project, funded by the European Union.

Ravi Kiran Sarvadevabhatla received the M.S. degree in computer science in 2006 from the University of Washington, where he was a member of the Laboratory for Neural Systems. He is currently a research engineer at the Honda Research Institute USA in Mountain View, California. His previous research includes indoor mobile robot localization, outdoor terrain traversability prediction for wheeled mobile robots, and object recognition for assisting brain–computer interface based control of humanoid robots. His current research interests are applications of computer vision for activity monitoring and guiding cognitive attention for Honda’s ASIMO humanoid robot.

Joel A. Wormer received the bachelor of science degree in computer science from the University of Illinois, Urbana, Illinois in 1979. He is currently a senior software architect at the Honda Research Institute USA in Mountain View, California. Prior to joining Honda, he worked in industry for 25 years, specializing in distributed control systems and distributed information systems. His current research interests include distributed architectures for robotics systems and hierarchical composition of robot behavior.

Thor List holds a bachelor’s degree in physics and computer science from the University of Copenhagen and a master’s degree in astrophysics. He expects to receive his Ph.D. in artificial intelligence from the University of Edinburgh in 2009. He also held the position of research fellow at the Institute for Perception, Action and Behaviour at Edinburgh University. He is an experienced software designer and developer with a decade and a half of commercial experience from major European and U.S. companies. In the last 10 years, he was CTO of CMLabs Inc. in New York, Head of Technology for Orcawise in Dublin and Chief Scientist for Pointstar in Denmark. His speciality is designing and developing large-scale, cross-platform software solutions.

Address for Correspondence: Victor Ng-Thow-Hing, Honda Research Institute, 800 California St., Sutte 300, Mountainview, CA 94041, USA. E-mail: vng@digital-monkeys.com.