DIARC: A Testbed for Natural Human-Robot Interaction

P. Schermerhorn, J. Kramer, T. Brick, D. Anderson, A. Dingler, and **M. Scheutz**
Artificial Intelligence and Robotics Laboratory
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
Email: {pscherm1,jkramer3,tbrick,danderso,adingler,mscheutz}@cse.nd.edu

Abstract

DIARC, a *distributed integrated affect*, *reflection*, *cognition* architecture for robots, provides many features that are critical to successful *natural human-robot interaction*. As such, DIARC is an ideal platform for experimentation in HRI. In this paper we describe the architecture and and its implementation in ADE, paying particular attention to its interaction capabilities and features that allow robust operation. These features are evaluated in the context of the 2006 AAAI Robot Competition.

Introduction

Autonomous human-like robots that interact in natural language with people in real-time pose many design challenges, from the functional organization of the robotic architecture, to the computational infrastructure possibly employing middle-ware for distributed computing, to the hardware operating many specialized devices for sensory and effector processing in addition to embedded controllers and standard computational boards. The task is to achieve a functional integration of very diverse modules that operate at different temporal scales using different representations on parallel hardware in a reliable and fault-tolerant manner that allows for *natural, believable human-robot interaction* (HRI). To achieve reliable, natural interaction with humans, several challenging requirements must be met, two of which are (R1) appropriate interaction capabilities, including *natural language capacity* (speech recognition and speech production), *dialog structure* (knowledge about dialogs, teleological discourse, etc.), *affect recognition and expression* (both for speech as well as facial expressions), and mechanisms for *non-verbal communication* (via gestures, head movements, gaze, etc.); and (R2) mechanisms for ensuring robust interactions, including *recovery from various communication failures* (acoustic, syntactic, semantic misunderstandings, dialog failures, etc.) as well as *software and hardware failure recovery* (crashes of components, internal timing problems, faulty hardware, etc.).

We are developing DIARC, a *distributed integrated affect*, *reflection*, *cognition* architecture for robots that interact naturally with humans (Scheutz *et al.* 2005; Scheutz, Schermerhorn, & Kramer 2006). We use our implementation of DIARC as a testbed for research on natural human-robot interaction. DIARC provides several of the features for the study of natural human interaction described above that are not easily found in other robotic systems. Some of these features were demonstrated in the 2006 AAAI Robot Competition and Exhibition. This paper provides an overview of the architecture as implemented for Rudy (our robot competition entry; see Figure 1) and of our experiences with Rudy at the competition. In Section , we describe different components of DIARC that are critical for HRI. Section recounts the system's performance at the competition. The paper ends with a brief account of related work in Section and some concluding thoughts in Section .

DIARC Implementation

DIARC is implemented in ADE, a Java-based infrastructure for the design, development, and execution of agent architectures (Scheutz 2006). To address requirement (R2), ADE provides robust, reliable, fault-tolerant middle-ware services for the distribution of complex robotic architectures over multiple computers. The ADE infrastructure provides the means of implementing an abstract functional architecture as a multi-agent system (MAS), where architectural components assume the role of *agents* that together form an integrated agent architecture. In ADE, each functional component is an `ADEServer`, with the ability to provide and/or make use of *services* when connected with other components. In addition to mechanisms that support component distribution and parallel operation, the framework also supplies monitoring, failure detection, and failure recovery services that tie directly into the high-level action interpreter.

A concrete example of DIARC's implementation across three hosts is shown in the "3-level" diagram in Figure 2. The top level is the "Abstract Agent Architecture" (i.e., DIARC), where darkened ovals represent functional architecture components, shown in a data flow progression from sensory input on the left, to data processing in the middle, and effector output on the right. The middle, or "ADE Component" level, depicts `ADEServer`s and their connections; each rounded rectangle is an `ADEServer`, dotted lines represent *heartbeat*-only connections and dot-dash lines represent combination heartbeat/data transfer connections. Solid and empty arrowheads indicate a component is either providing or using services, respectively. Hardware devices

Figure 1: The robot at the 2006 AAAI Robot Competition.

used by an ADEServer are depicted by a set of labeled squares within the server's rectangle. The bottom, or "Hardware" level, specifies hosts and their connections. Two relations between the bottom and middle levels are shown: (1) ADEServers are placed in a vertical column directly above the host on which they execute and (2) connections between hardware devices and the ADEServers that use them are indicated by solid lines that cross the level divider. The relation between the middle and top levels is represented by mapping darkened ovals (i.e., functional architectural components) to the ADEServer that implements them.

ADEServers are not necessarily part of the functional architecture, however. For instance, the ADERegistry, a special type of ADEServer, is an infrastructure agent that mediates connections among ADEServers and the processes that use their services. In particular, an ADERegistry organizes, tracks, and controls access to ADEServers that register with it, acting in a role similar to a MAS *white-pages service*. During registration, an ADEServer provides information about itself to the ADERegistry, also establishing a heartbeat that periodically updates the server's status. Thus, an ADERegistry forms a gateway into an ADE configuration; to gain a connection with another server, an ADEServer must make a request to an ADERegistry. Upon approval and successful connection, a separate heartbeat is established between the servers.

Heartbeats provide the foundation for failure detection and recovery that can be exploited to build rudimentary "self-awareness" into the system. Both ends of the connection monitor the heartbeat: the sending component receives an error if it cannot "check-in", while the receiving component periodically confirms heartbeat reception and times out if none has arrived. A non-registry ADEServer uses heartbeat signals to determine the status of *connected* components; that is, whether the services of an external component remain available or whether *reconnection* is required, possibly altering normal function or relaying notice of the failure to other connected components. An ADERegistry, on the other hand, uses this information to determine the status of server *operation*; that is, whether a server exists and is executing properly or whether the server needs *recovery*.

For instance, referring to Figure 2, the "Action Interpreter and Excecution" component (left column) is connected to both the "Speech Recognition" server (right column) and the "Robot" server (middle column). Heartbeats sent from the action interpreter confirm the availability of services, even when they are not being used. All three servers are registered with the ADERegistry (center column), sending heartbeats that indicate their continued operation. During execution, verbal commands from the user that are recognized (e.g., "Move forward") are interpreted and result in a motor command sent to the robot. Now consider the case where "Laptop-2" is disconnected from the network: the speech recognition component fails, thereby rendering the robot unable to understand commands. The interpreter will be unable to send its heartbeat, triggering attempts at reconnection and, for safety reasons, reacting to the error by sending a command to the robot that stops the motors. At the same time, the ADERegistry detects the failed update and can initiate diagnostic and/or recovery procedures (which may result in migration of the component to another host). Upon recovery of the speech recognition component, it re-registers and once again becomes available; the inter-

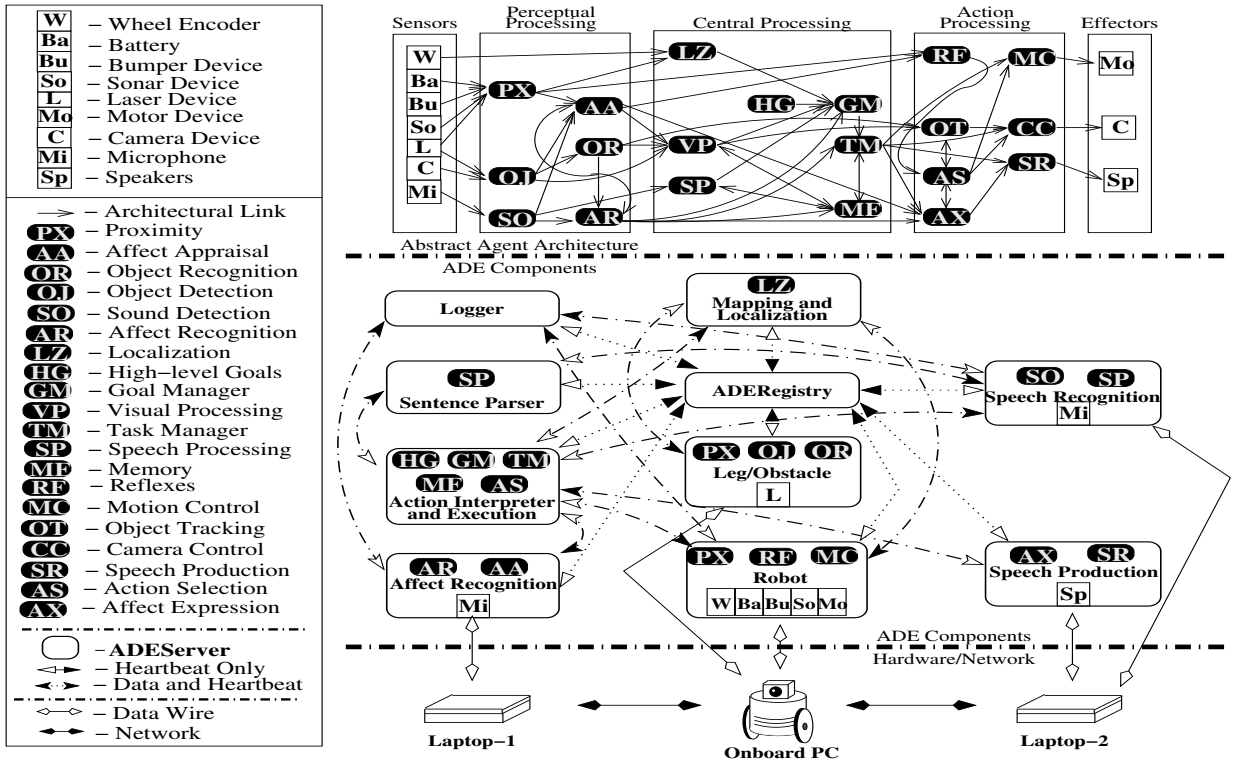

Figure 2: A depiction of the DIARC architecture as implemented in ADE, distributed across three hosts.

preter automatically reconnects, turns the motors back on, and the system is restored to full operation.

A novel feature of ADE is a *reflective reasoning* component that is incorporated directly into the infrastructure to enhance the capabilities outlined above. In particular, the reasoning component allows the system to maintain a knowledge base of "facts" about itself (e.g., active and uninstantiated components, relationships among components, known hosts and their properties, etc.). Furthermore, the knowledge base can include rules (i.e., a "policy") that refine the reactions (e.g., reconnection and recovery) to changing conditions; for instance, allowing an `ADERegistry` to use the facts to reason about the recovery process. The registry can consider the properties of the known hosts for component relocation suitability. Potential hosts that cannot support a component due to unsupported hardware requirements are filtered out and the remaining possibilities are ordered using the facts about the hosts and some measure of preference (e.g., the host with the fastest processor, lowest CPU load, or some combination of qualities). Continuing the example above, the registry can determine that speech cannot be recovered on the robot, which lacks a microphone and so relocates it to "Laptop-1". Now assume that the network interruption was temporary; even upon restoration, the policy may determine that the failed component should be recovered on a different host, e.g., if the system load of "Laptop-1" is less than that of "Laptop-2".

Thus far, only infrastructure mechanisms have been described. However, the infrastructure can also be closely integrated with the functional architecture. One example, given above, is the potential of one component to relay information to another (the interpreter shutting off the robot's motors). Another possibility is to use an alternate component that switches *modalities*; rather than require speech recognition, the action interpreter may determine that keyboard input is sufficient and recovery may consist of starting a "Keyboard Input" server (not shown in the figure). An even more integrated response might be the instantiation of a completely new goal–the interpreter may decide that its current task is no longer feasible and choose a totally different goal (e.g., go to the robot maintenance area for repair).

Perceptual Learning

The visual perception system is composed of two main components: a long-term memory and a short-term memory. The long-term memory system allows the robot to associate the scale-invariant feature transformation (SIFT) keypoints of an object with a specified token, thus learning the object. Unlike many learning techniques, including Haar cascades, SIFT keypoints can be learned from a single training image, in a relatively short period of time (on the order of a few seconds, depending upon the resolution of the image and the available computing power). The ease and speed of training allows the robot to learn arbitrary objects during runtime. This is accomplished in the following manner: an individual holds up an object to the camera and specifies a token

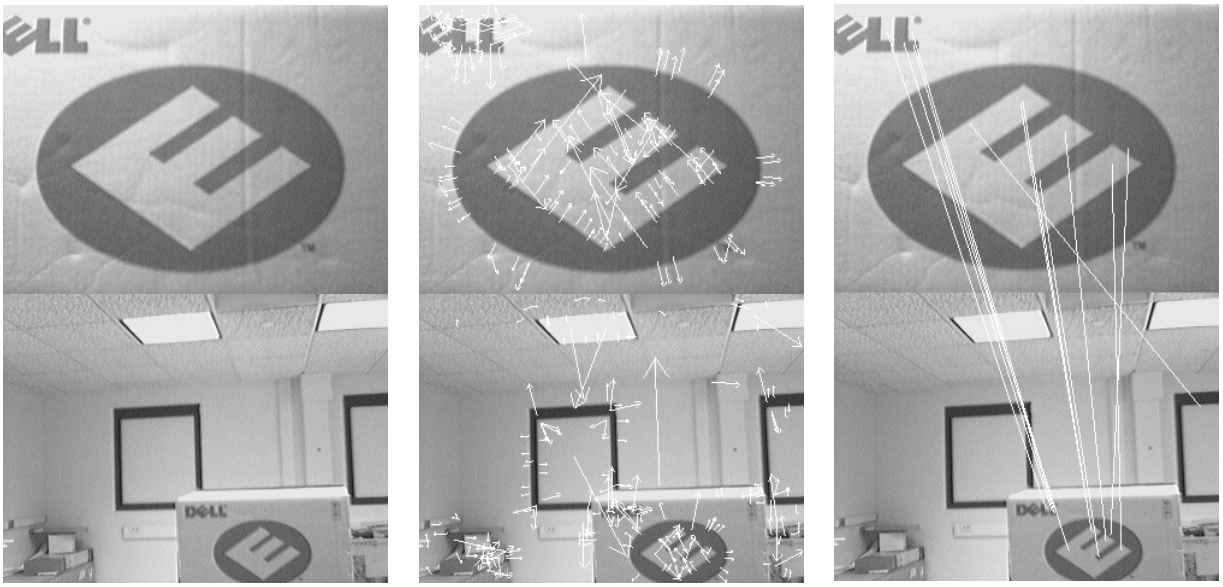

Figure 3: **Left:** Unprocessed training (top) and scene (bottom) images. **Center:** SIFT points located during individual processing. **Right:** SIFT point matches detected when comparing the two images.

(with help from the voice natural language processing system), the visual system then determines the SIFT keypoints of the frame and adds an entry into a database, which associates the token with the keypoints, allowing for subsequent retrieval. This database serves as the system's *visual long-term memory*, as it can be saved and restored on subsequent runs, allowing the system to "remember" objects it has identified during earlier interactions.

The other component of the vision system's two-level storage scheme, *visual short-term memory*, is used as working memory for currently or recently perceived objects. In visual short-term memory, data from the laser rangefinder and two cameras are combined to build a model of the people and objects surrounding the robot. The system is able to detect and track people as they move about in the environment. The location of a person is determined by combining a pair of legs (determined by the laser rangefinder) with a face. Face detection is performed using the combination of a hierarchical swarm system and a Haar cascade. The Haar classifier is used to constrain the region in which the swarm search is performed (i.e., only look for facial features in face regions). The lower level swarms detect facial features (specifically two eyebrows and a mouth), and the higher level swarms define the geometric relationship between these features. The combination of the two techniques allows for a higher framerate (as the Haar cascade update need not be performed every cycle) and better tracking (as the swarms are less sensitive to changes in facial orientation).

Object recognition is performed by capturing a frame of the scene, determining its SIFT keypoints and the performing a least-squared regression between the object(s) in the database (see Figure 3). If a sufficient number of keypoints are found between an object in the database and the frame just captured, it is possible to conclude with a fair degree of certainty that the robot is "seeing" the object. The system is thus able to answer queries (again with the help of the natural language subsystem) such as: "Do you see an X?" and "What do you see?" Since the robot has two cameras, it is possible to determine the three-dimensional location location of such objects (combined with the height of the robot, the position of the pan-tilt unit and the distance between the two cameras). While this position is not extremely precise, it is sufficient for the system to determine spatial relationships between the objects. This allows the robot to answer more interesting queries, such as: "Is X to the left of Y?" or "What is the spatial relationship between X and Y?" Finally, the system is able to answer queries from the perspective of another person or object in the robot's short-term memory. This allows the robot to answer questions such as: "From the perspective of X, is Y to the left of Z?"

Incremental Natural Language Processing

The robot interacts with humans by using spoken natural language in the course of executing conversation and action scripts. The *natural language processing subsystem* uses a novel *Parallel Incremental Processing* (PIP) architecture to interpret the phonetic, syntactic, and semantic levels of utterances spoken. In this architecture, each word in an utterance is phonetically interpreted and then simultaneously syntactically analyzed and semantically understood as soon as it is available to the system.

The interaction between the action interpreter and the PIP natural language module is performed using the getBindingOfType method. The action interpreter, based on its current state, requests a specific type of binding from the discourse model. In our Planetary Exploration Interaction, this is most commonly a binding of type "action". The discourse model

then seeks to interpret incoming utterances with the intent of finding an action to be executed. The action interpreter can then later check to see if the request has been fulfilled, and to retrieve the resulting binding. This mechanism could be used in other ways, however. For example, in a "robot bartender" scenario, the action manager could request a binding of type "drink", in the hopes of retrieving a drink request from its customer. It is important to note that several binding requests can be maintained simultaneously, so a request for a drink would not invalidate any attempts to give the robot commands, for example.

The first level of interpretation within the natural language processing subsystem is performed by a modified version of the CMU Sphinx4 natural language interpreter, which translates the phonetic utterance, word-by-word, into strings of words recognizable by the incremental discourse model. These translations are limited by the addition of a regular grammar, which is put into place by the action interpreter when an interaction is started. The addition of this grammar allows the natural language processing subsystem to "force" an utterance to be heard as an appropriate string of words, and to ignore utterances that are not in the grammar, significantly increasing the understanding of terms in a noisy environment. Furthermore, the grammar is easily and quickly changed during runtime, so if the robot finds that it is entering a new context it can change the grammar accordingly in order to improve recognition and understanding.

Once a word is phonetically determined, a structure is retrieved from memory containing possible semantic meanings of the words and their related syntactic constraints. These constraints are similar in nature to the synsets in WordNet (Miller 1995) and verb senses in VerbNet(Kipper, Dang, & Palmer 2005), but are specifically tailored to the action primitives used by the action interpreter. That is, the robot's understanding of the semantic meaning of a command is the action interpreter primitive associated with that command.

As each word is added, the syntactic and semantic constraints associated with each of its possible semantic meanings are compared with the constraints of those sentence interpretations that are still currently viable. Each later word then selects only those sets of its own syntactic and semantic constraints which do not conflict with the existing interpretations, and additionally removes those possible interpretations which conflict with all of its own possible interpretations.

We include below (edited for space) a trace of the system processing the phrase "move to the left". Different syntactic/semantic possibilities are separated vertically, connected by a vertical pipe ("|"). Parentheses ("()") indicate optional but likely additions, and triangle brackets ("<>") indicate the syntactic role of future arguments.

As the word 'move' is understood by the system, it generates three semantic/syntactic frames. The first is associated with the action manager script "startmove¡direction¿:?extent", and requires a following direction and an optional extent. The second is associated with the action script "startmove:¡location¿", and requires a preposition and a location. As more words are processed, the system further and further constrains the possible options. When the utterance ends, only one possible interpretation remains, so the action manager is passed that action command.

```
 Word Processed: move, type Verb. Constraints:
 |Move (<preposition:destination> <article:definite>)
 |     <direction>(<extent>)
 | (startMove<direction>:?extent)
 |
 |Move <preposition:destination> <location>
 | (startMove:<location>)
 |
 |Move <object> <preposition:destination> <location>
 | (moveObject:<location>)

Possible Actions:
  startMove, moveObject

Word Processed: to, type preposition:destination.
Constraints:
|Move to <article:definite> <direction> (<extent>)
| (startMove<direction>:?extent)
|
|Move to <location>
| (startMove:<location>)
Confirmed Action: startMove

Word Processed: the, type article:definite.
Constraints:
|Move to the <direction> (<extent>)
| (startMove<direction>:?extent)
|
|Move to [the <noun:inanimate>]
| (startMove:<location>)
Confirmed Action: startMove

Word Processed: left, type direction modifier.
Constraints:
|Move to the left (<extent>)
| (startMoveLeft:?extent)
Confirmed Action: startMove(Left)
```

The interpretations generated by the system are all ranked by expected likelihood, so that there is always a "most likely" interpretation (the top one in the trace) of a given utterance. If more than one possible interpretation is available by the end of the utterance, the system either chooses the most likely interpretation, or generates an appropriate question.

Similarly, if only a single interpretation of the words is found by the end of the utterance, but a constraint is still unfulfilled at that point, the system generates a question requesting that type. For example, the word "look" generates only one semantic/syntactic interpretation. It has the meaning "look:¡direction¿?extent" (extent optional). If no words follow that indicate a direction, the system will generate the contextual question "which direction?". A binding of type "direction" will then be requested through the existing getBindingOfType mechanism. If the following utterance is simply "left", it will be recognized as a direction binding, and passed back to fill the required slot, allowing the robot to begin its look. Because binding requests do not interfere with one another, a response of "look right" will still be recognized normally. In this case, the temporary binding

request for a "direction" will be invalidated, since its context is no longer valid.

The Goal and Task Managers

Action selection and execution in this implementation of DIARC are performed by the Action Manager and the Action Interpreters it controls. The Action Manager's role is to instantiate new processes when goals are added and mediate between goals when conflicts arise. Each new high-level goal is assigned to an Action Interpreter, which executes a script that satisfies that goal. Scripts can be viewed both as encoding situational knowledge of how certain common interactions tend to proceed, and as encoding procedural knowledge that the robot can employ as steps in a plan. Various control constructs (e.g., conditionals, loops, etc.) are available to script writers. This allows the system to, for example, branch on the success or failure of an action to react appropriately. Scripts can call other scripts (i.e., instantiate subgoals and and perform the actions required to satisfy them), which execute in the same context as the parent script (i.e., a script invocation does not lead to the instantiation of a new Action Interpreter).

As a trivial example, the following script (activated from a higher-level script as `move-to-transmit self data location3`) instructs the robot to return to the transmission location `location3` (previously identified and stored in a map as part of long-term memory), check to make sure the signal strength remains sufficient for successful transmission, and transmit the data:

```
move-to-transmit        ; transmit data at xmit-pt
    robot
    data
    xmit-pt
move-to robot xmit-pt   ; go to location xmit-pt
verify-signal robot     ; is signal strong enough?
transmit robot data     ; transmit data
```

The action interpreter substitutes the arguments passed for each of the roles in the script and begins executing the first event. A behavioral primitive like `move-to` then has a particular meaning to the robotic system. In this case, the action interpreter passes the action on to the navigation system, which interprets it as a command to move the robot to the coordinates (x, y) of `xmit-pt`. The high-level navigation system generates a plan which translates the action into commands for the low-level navigation system, eventually causing the robot to move in a particular direction, if possible (e.g., it will not move there if obstacles block the location, although it will attempt to move around obstacles that obstruct the path to the final location).

Failure recovery actions can be encoded directly in the scripts. For example, if the signal strength at the transmission point no longer exceeds the minimum threshold, transmitting is pointless (and may be costly). In the script above, `verify-signal` fails, which in turn causes `move-to-transmit` to fail, avoiding the `transmit` step. The calling script can detect the failure of `move-to-transmit` and react appropriately (e.g., finding a human to ask for further instructions, or initiating a search for another transmission point).

The Action Manager periodically updates the *priority* of each goal. These goal priorities are used to determine the outcome of conflicts between Action Interpreters (e.g., resource conflicts, such as when each wants to move in a different direction). A goal's priority is determined by two components: the *importance* and the *urgency*. The importance of a goal is determined by the cost and benefit of satisfying the goal. This utility is scaled by the urgency component, which is a reflection of the time remaining within which to satisfy the goal:

$$Urg = \frac{Time_{elapsed}}{Time_{allowed}} \times (Urg_{max} - Urg_{min}) + Urg_{min}, \quad (1)$$

where Urg_{max} and Urg_{min} are upper and lower bounds on the urgency of the goal. The goal's priority, then, is simply:

$$Priority = (Benefit - Cost) \times Urg. \quad (2)$$

This formulation allows goals of lower importance, which would normally be excluded from execution in virtue of their interference with the satisfaction of more important goals, to be "worked in" ahead of the more important goals, so long as the interrupted goal has sufficient time to satisfy the goal after the less important goal completes (i.e., so long as the urgency of the more important goal is sufficiently low).

The Robot at the Competition

The system described above is well-suited to perform in many of the categories of the Human-Robot Interaction Competition. However, due to hardware failures (see below), we were able to successfully participate in only two of the seven: Category 3 (Natural language understanding and action execution) and Category 4 (Perceptual learning through human teaching and subsequent recognition and categorization of people, objects, locations or actions).[1] This section describes the robot's performance for each of these tasks.

Natural language understanding and action execution. This category includes following requests from humans to move in particular ways; getting a requested item from some other person; understanding descriptions of directions and applying them to lead other people to specific places; etc. It was here that we had the most success, having been awarded the technical award for this category.

The natural language processing and action execution modules of the robot performed approximately at the level of expectation, with one primary failure: high levels of human-voice noise defeated our noise-reduction attempts, and made it difficult for the robot to perform even simple natural language tasks on the exhibition floor. We were able to mitigate this difficulty to some extent by forcing the robot to interpret each sound it heard as the closest reasonable utterance within its regular grammar, but this unfortunately an

[1] See http://www.nd.edu/~mscheutz/humanrobotinteraction.html for an overview of all categories.

increase in the frequency with which the robot would incorrectly classify noise as a valid utterance. In environments with relatively low human-voice noise, the robot was able to recognize commands via human speech, interpret them, and carry out an appropriate action.

Many of the collaborative tasks prepared for the competition involved movement on the part of the robot. However, a catastrophic failure of the robot base left the robot without onboard computer or drive motors. This failure limited the collaborative abilities the robot could demonstrate to head movements and verbal responses to questions from its perceptual learning engine. While it performed reasonably well at this limited set of tasks, the overall demonstration was not representative of the robot's total skill set.

Specifically, a demonstration was prepared of a collaborative exploration task in which the human team member would direct the robot to locate optimal transmission locations, as indicated in a map of simulated "signal strengths" detectable only to the robot. This collaboration required the robot to understand natural spoken language and respond appropriately (e.g., reporting the local signal strength or moving in the direction indicated). In parallel with this collaborative task, the robot would independently pursue a goal of locating certain (previously learned) objects placed in the environment. The object here was to catalog their locations, demonstrating the robot's ability to perform multiple tasks concurrently.

Perceptual learning through human teaching and subsequent recognition and categorization of people, objects, locations or actions. This category includes remembering the face of a person; learning of a location in the environment and being able to remember and recognize it; learning what it means to "turn around" and being able to repeat it; learning what an object like a Coke can looks like and recognizing it among other different objects; etc.

Our proposed perceptual learning system performed rather well at the conference. It was able to visually detect and track (using dual fire-wire cameras and a pan-tilt unit) a number of people in its surroundings. It had some difficulty performing this behavior in extremely crowded surroundings, though this could likely be improved by augmenting the leg tracking code. It was able to learn objects and identify them with a relatively high success rate. Most of the shortcomings in this area were due to problems with the SIFT algorithm. SIFT works very well on objects with a high number of distinguishable characteristics. However, it has trouble on more "plain" objects, or objects that have few easily distinguishable patterns (e.g., angles). The system also had some trouble with objects that were so far away they lacked detail, due to lack of camera resolution. The integration of the visual system with the other systems, particularly the natural language recogntion system, worked well and provided an effective interface for interacting with the perceptual learning subsystem.

Related Work

Spartacus (Michaud *et al.* 2005), the Laborius project's entry in the Human-Robot Interaction event, handled the unstructured environment of the conference quite gracefully. Most impressively, the speech recognition issues that continually challenge our system, Rudy, are non-issues for Spartacus. The ability to extract voices even in noisy conditions is essential to natural HRI, and will need to be addressed in our platform.

Washington University's Lewis (Smart *et al.* 2003) incorporates many features similar to those of our system. In particular, Lewis' software framework includes mechanisms for graceful failure recovery, similar in some ways to the infrastructure mechanisms for recovery provided by ADE. In addition, some of the experiments being conducted using Lewis are similar in spirit to some conducted in our own lab. Lewis has been used for experiments based on a scavenger hunt in which a human subject interacts with the robot to locate objects of interest (Heckel & Smart 2006); this is similar in many ways to a planetary exploration task used in some of our experiments, although at this point Lewis is teleoperated for their experiments.

Grace and George (Gockley *et al.* 2004) and Hermes (Bischoff & Graefe 2003) are probably the closest to our project in terms of being capable of natural language interactions with humans and integrating higher level deliberative components in the underlying distributed implementation of the robotic architecture (e.g., with respect to deliberative modules, action interpretation, etc. the modular design of Hermes is closer in spirit to our proposed MAS system than the somewhat "adhoc" integration of components in Grace and George). Yet, both architectures lack the intrinsic involvement of affect mechanisms in the architecture and many of the infrastructure mechanisms provided by a MAS like ADE, both of which are characteristic features of DIARC.

Conclusion

DIARC supplies a variety of interaction capabilities and mechanisms to ensure robust system operation, making it a good choice as an experimental testbed for research into human-robot interaction. Although hardware failures prevented the robot from demonstrating all of its abilities, successful participation in the "natural language understanding and action execution" and "perceptual learning" categories of the Human-Robot Interaction event at the 2006 AAAI Robot Competition even in the face of these catastrophic failures serves as evidence of DIARC's promise as an architecture for HRI. We have successfully used the architecture for a variety of experiments with human subjects, albeit in the controlled environment of the laboratory. Although a great deal more work is required before any present architecture is able to provide natural human-robot interaction ability, DIARC is a start at addressing some of the critical issues facing researchers in HRI.

References

Bischoff, R., and Graefe, V. 2003. Hermes – an intelligent humanoid robot, designed and tested for dependability. In *Experimental Robotics VIII, Proceedings of the 8th International Symposium ISER02*.

Gockley, R.; Simmons, R.; Wang, J.; Busquets, D.; DiSalvo, C.; Caffrey, K.; Rosenthal, S.; Mink, J.; Thomas, S.; Adams, W.; Lauducci, T.; Bugajska, M.; Perzanowski, D.; and Schultz, A. 2004. Grace and george: Social robots at aaai. In *AAAI 2004 Mobile Robot Competition Workshop (Technical Report WS-04-11)*.

Heckel, F., and Smart, W. D. 2006. Non-speech aural communication for robots. In *to appear in Proceedings of the 2006 AAAI Fall Symposium*.

Kipper, K.; Dang, H.; and Palmer, M. 2005. Class-based construction of a verb lexicon. In *Proceedings of AAAI 2000*.

Michaud, F.; Brosseau, Y.; Ct, C.; Ltourneau, D.; Moisan, P.; Ponchon, A.; Raevsky, C.; Valin, J.-M.; Neaudry, .; and Kabanza, F. 2005. Modularity and integration in the design of a socially interactive robot. In *Proceedings IEEE International Workshop on Robot and Human Interactive Communication*.

Miller, G. 1995. Wordnet: A lexical database. *Communications of the ACM* 38(11).

Scheutz, M.; Schermerhorn, P.; Middendorff, C.; Kramer, J.; Anderson, D.; and Dingler, A. 2005. Toward affective cognitive robots for human-robot interaction. In *AAAI 2005 Robot Workshop*.

Scheutz, M.; Schermerhorn, P.; and Kramer, J. 2006. The utility of affect expression in natural language interactions in joint human-robot tasks. In *Proceedings of the IEEE/ACM 1st Annual Conference on Human-Robot Interaction*.

Scheutz, M. 2006. ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence* 20(4-5).

Smart, W. D.; Dixon, M.; Melchior, N.; Tucek, J.; and Srinivas, A. 2003. Lewis the graduate student: An entry in the aaai robot challenge. In *AAAI Mobile Robot Competition 2003: Papers from the AAAI Workshop*.