International Journal of Semantic Computing
© World Scientific Publishing Company

NATURAL LANGUAGE INTERACTIONS IN DISTRIBUTED NETWORKS OF SMART DEVICES

PAUL SCHERMERHORN and MATTHIAS SCHEUTZ
Human-Robot Interaction Lab, Indiana University
Bloomington, IN 47406, USA
{*pscherme,mscheutz*}*@indiana.edu*
http://hri.cogs.indiana.edu

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Advances in sensing and networking hardware have made the prospect of ambient intelligence more realistic, but the challenge of creating a software framework suitable for ambient intelligence systems remains. We present ADE, the *Agent Development Environment*, a distributed agent infrastructure with built-in natural language processing capabilities connected to a sophisticated goal manager that controls access to the world via multiple server interfaces for sensing and actuating devices. Unlike other ambient intelligence infrastructures, ADE includes support for multiple autonomous robots integrated into the system. ADE allows developers of ambient intelligence environments to implement agents of varying complexity to meet the varying requirements of each scenario, and it provides facilities to ensure security and fault tolerance for distributed computing. Natural language processing is conducted incrementally, as utterances are acquired, allowing fast, accurate responses from system agents. We describe ADE and a sample of the many experiments and demonstrations conducted using the infrastructure, then an example architecture for a "smart home" is proposed to demonstrate ADE's utility as an infrastructure for ambient intelligence.

Keywords: Ambient Intelligence; Natural Language Processing; Robotics.

1. Introduction

Imagine the following scenario: you return home after a long day at work, and walk into your house, a "smart home" equipped with ambient intelligence where different software "agents" represent different parts of the house. As you enter the house, the `hallway agent` recognizes you (based on visual, auditory, and other cues) and greets you ("Hello Dave, I hope you had a good day at work."). The agent begins tracking your movements through the house (using a multi-modal "person tracking" system) to determine where you will go first. Once it becomes clear that you are heading to the living room, the `living room agent` consults a knowledge base (for your previous actions when going to the living room upon returning home) and determines that you are likely to want to watch the news while the `kitchen` `agent` begins heating the evening's dinner, so it passes that information on to the `kitchen agent`. The `living room agent` turns the lights and television on and changes the channel to the one with your favorite news show (again, based on information stored in the knowledge base). Meanwhile, the `kitchen agent` asks you what you would like for dinner. Once you express your preference, the `kitchen agent` verifies that the appropriate food is on hand and begins preparations. Then you decide you'd like a drink: "I'd like a scotch, please." The `living room agent` handles the request, but also relays it to the `kitchen agent`, which, based on past experience, knows that it is likely that you will want dinner delayed (had you asked for wine, dinner preparations would have continued uninterrupted). The `kitchen agent` asks, "Dave, would you like me to hold off on heating dinner for a while?" You agree that a half-hour delay would be nice and settle in to watch the news, putting dinner out of your mind until a little over 30 minutes later when the `kitchen agent` informs you that the meal is ready.

This scenario could be dismissed as straight from science fiction, requiring a great deal of work in multiple areas to approach reality. However, the feasibility of ambient intelligence has been steadily increasing especially with advances in hardware [12]. Miniaturization of sensors and actuators, as well as the increasing presence of smart devices, for example, provide opportunities for "ubiquitous computing" environments. Embedded computers can be found in virtually every area of the modern household, from washing machines to kitchen appliances, televisions to automobiles. Similarly, advances in networking, perhaps especially wireless technologies (e.g., Crossbow and ZigBee devices), open the possibility of bringing all these elements together into a coherent system capable of responding to and even anticipating users' needs and desires in a variety of settings. Hence, it is clear that sensing and networking technology has already advanced to the point at which the scenario above could be realized.

What about software technology? Clearly, advances need to be made in the artificial intelligence for the reasoning processes described in this scenario. Approaches such as case-based reasoning [22] attempt to bridge that gap—the present work does not claim to solve the reasoning problem. However, in addition to the reasoning problem, two important characteristics of the "smart home" in this scenario stand out: that various components are tightly interconnected by an advanced distributed computing infrastructure, and that most interaction with the human is done via natural language dialogue. Hence, to realize the smart home scenario, we need a software infrastructure that combines distributed computing with advanced natural language processing capabilities. There are a number of projects focused on addressing the software needs of ambient intelligence and ubiquitous computing (e.g., [16, 11, 4]), but none of these attempts to integrate NLP in a way suitable for ambient semantic computing. Other projects focus on multi-modal natural language interactions [14, 13], however, they do not provide support for integration of robotic agents in the architecture, an addition that will only become more important as the presence of household robots increases. We present here the *Agent*

Development Environment (ADE), a software infrastructure package that provides the connectivity and fault-tolerance required for ambient intelligence and features integrated incremental NLP and other components required for complex tasks and natural language interactions with humans. In the rest of the paper, we will provide an overview of ADE and how it can be used to support ambient semantic intelligence. Section 2 provides background, motivating the need for sophisticated software support for ambient intelligence. In Section 3, we detail the requirements that must be met to support ambient intelligence and discuss how ADE addresses them. An example architecture is presented in Section 4. The paper closes with some discussion in Section 5 and concluding remarks in Section 6.

2. Background

One motivating factor behind the smart home concept is the desire to be able to assign simple tasks to "the system" that would normally require at least minor intervention on the part of the user. Hence, many components of the smart home architecture will have actuating capabilities (e.g., changing the temperature, selecting a CD to play) although some will be simply sensors. While most agents in an ambient intelligence system are stationary (again, think of a home with multiple smart appliances and other devices), support for mobile agents will allow for useful expansion of system capabilities [25]. Mobile robots could serve as room cleaners (e.g., dispatching the sweeper-bot to vacuum the floor), monitors for the elderly (allowing for increased autonomy while still keeping an eye on vulnerable persons), or the ever-popular "robot waiter," bringing a drink from the refrigerator [30, 23].

The software required to realize ambient intelligence systems must provide a solid infrastructure of mechanisms for programming and controlling large, complex networks of heterogeneous devices (e.g., [1]). The infrastructure must enforce security and privacy policies if users are to trust the system to operate sensitive aspects of their homes; the system is likely to have access to a great deal of personal information about the user, including preferences (e.g., for movies or TV shows), history (e.g., food and drink requests), and passwords for accessing remote resources. Audio and video streaming capabilities also raise security concerns with the possibility of eavesdropping by tapping into the streams. The infrastructure also must implement mechanisms for fault tolerance, to ensure that the failure of some component in one area of the system does not cause cascading failures throughout the system. Finally, the system needs to allow for easy, intuitive interactions for control and inquiry; people are unlikely to use a system that provides a tedious, complex interface (e.g., requiring the user to be at a central location to interact via keyboard or mouse).

We believe that natural language interactions will thus constitute the core interface to complex systems such as "smart" homes, given that verbal communication comes most naturally to humans, despite the challenges NLP poses for computer systems. Some system components, such as software agents representing the interfaces to rooms in the above scenario, will have NLP capabilities (i.e., be able to respond to informational queries—"How many apples are there in the refrigerator?"—or initiate actions based on commands—"Please increase the volume by 10%"). In order to provide the level of responsiveness (e.g., confirming commands, or asking for clarification) that people expect in verbal interactions, incremental NLP, in which the intelligent agent begins processing the human's sentence before it is complete, will be essential [7]. Much of the time, human interlocutors have already determined the meaning of a spoken sentence even before it is complete; any delay for NLP after the utterance will detract from the naturalness of the interaction. Incremental NLP will also allow the agent to begin carrying out requests (e.g., "Could you lower the blinds, please") before the request has been completely uttered.

All these requirements seem to suggest a complex system that handles natural language processing, knows user preferences, and has a large set of skills that can be applied to useful real-world actions. The problem of designing such a system can be simplified by decomposing it into several more tractable subsystems that interact with each other. Instead of a single monolithic entity (a là the computer on Star Trek), a successful approach to ambient intelligence will decompose a system into multiple distributed components ("agents") with multiple intelligences of varying degrees of sophistication. The ambient intelligence system then is really a collection of these different "cognitive agents" of varying complexity combined with various low-level servers that do not have any "cognitive capabilities" *per se*. Different subsystems (e.g., different rooms in a house) are represented in the architecture by different entities that are specialized to the kinds of tasks and requests that are normally salient to their environments. For example, with regard to NLP, the problem of understanding can be greatly simplified by limiting the language domain to fit the scenario. Similarly, not all capabilities will be needed in all scenarios (e.g., the `bathroom agent` does not necessarily need to understand how to begin preparing dinner, it only needs to know how to pass the request on to an agent that does, the `kitchen agent`).[a] Finally, it is probable that different agent representations will be better-suited to different scenarios, and the multi-agent approach will facilitate this customization.

A number of projects have focused on one or more of the aspects described above. The SmartKom architecture provides infrastructure support for multi-modal (e.g., speech, gesture) dialogue, utilizing a large number of components to interact with humans [14]. SmartKom, based on the distributed integration platform MULTIPLATFORM [13], has been demonstrated in a variety of scenarios, including the SmartKom-Home interface to home entertainment systems [24], with functionality similar to the multimedia-related examples presented here (i.e., requests for information about upcoming shows, etc.). SmartKom is a mixed-initiative architecture that

[a]Clearly, the system must determine *where* to send the request, however the topic of the current paper is the infrastructure, not the reasoning ability. The infrastructure needs to provide mechanisms for forwarding messages when it is determined that such forwarding is required. How that determination is made is beyond the scope of this paper.

addresses many of the NLP-related challenges posed by the smart home scenario. The ADE infrastructure seeks to solve those problems, but also includes support for multiple autonomous robots integrated into the smart home architecture.

3. The ADE Infrastructure

In general, ambient intelligence systems will require infrastructures capable of supporting controlled access to different kinds of sensitive information (e.g., to prevent illicit access to personal preferences or data stored within the system or listening in to phone conversations, etc.), access to a variety of sensing devices, and interfaces for the effective control of actuating devices found in the system. Moreover, refined internal monitoring and supervision tools are needed to detect failures of components, initiate recovery from failure, and ensure the long-term autonomous operation of the system. Finally, for natural communication interactions with humans, sophisticated analysis and reasoning tools are needed that will integrate a great variety of current and future AI technology (from natural language processing and understanding, to machine translation, hypothetical reasoning with uncertainty, various forms of machine learning, and many others). What kind of system can provide the necessary infrastructure for the implementation of such an autonomous intelligent distributed application?

The natural place to look for an answer would be *multi-agent systems* (MAS), which are concerned with providing an agent-based infrastructure for distributed computing. Agents in the context of MAS, call them “MAS-agents,” are computational processes that implement the autonomous, communicating functionality of a distributed application [9]. While many MAS provide the necessary tools to implement secure, fault-tolerant distributed systems as determined by the “MAS-architecture” (i.e., the overall blueprint of the multi-agent system), they do not provide support for the implementation of functional components of the architectures of their constituent MAS-agents. Yet, support for components like memories, reasoning, planning, or learning engines, etc., is likely going to be necessary for the implementation of the many different kinds of tasks described above.

This kind of support is typically found in *single agent systems* (SAS), which focus on the organization of functional components in the “SAS-architecture” for intelligent agents. These intelligent agents, call them “SAS-agents,” are typically *situated* in some environment and are capable of *flexible autonomous* action in order to meet their design objectives [15]. However, SAS are not concerned with highly parallel processing environments with real-time requirements, having been largely developed for single CPU architectures. Hence, they do not provide the kinds of distribution mechanisms to preserve the parallelism that their architectures might allow for, nor do they provide mechanisms for controlled, secure access to parts of the architecture.

We believe that a *synthesis* of both kinds of systems is necessary, which can provide “computing infrastructure plus intelligence,” i.e., the support for SAS agent architecture development together with the infrastructure necessary for the interaction of MAS components. Our ADE framework for embodied real-time systems has been specifically developed as a MAS system infrastructure that allows for "MAS-agents" to be designed with "SAS-architectures" (e.g., the cognitive architecture SOAR [21]), where the components of these "SAS architectures" (e.g., rule-base, working memory, etc.) are themselves "MAS-agents." The main utility of such an infrastructure for ambient intelligence systems is that it allows (1) for a secure, fault-tolerant, large-scale distribution of the system at the level of architectural components of SAS-agents, (2) for the implementation of sophisticated reasoning tools in existing cognitive frameworks, and (3) for features resulting from integrating SAS and MAS that will significantly improve the long-term operation and maintenance of such systems.

To be able to distribute architectural components and implement them as MAS-agents, ADE, the *Agent Development Environment* [2], provides a basic network infrastructure of connected *ADE server objects*, with each server able to obtain references to other servers in the system that serve as remote representations of the resources offered by the remote server. The client-server subsystem is then used to implement various kinds of ADE *MAS-agents*, i.e., autonomous computational processes consisting of one or more clients and/or a server that communicate with other MAS-agents in the system (via client-server connections). The base *ADE server* provides the necessary run-time infrastructure and environment for derived ADE server MAS-agents, which implement the interfaces to the sensors, actuators, and computational resources of the ambient intelligence system. ADE defines several special MAS-agents as part of the MAS-architecture of an ADE system. For example, there are *ADE registry* MAS-agents that implement a system-wide yellow pages service, allowing other MAS-agents to register and advertise their services.

The ADE server model was designed for potentially vulnerable distributed dynamic multi-OS computing environments, where connections between any two hosts participating in the ADE system cannot be assumed to be secure nor present throughout the lifetime of an application. Consequently, mechanisms are required for ADE servers to protect communicated information, detect failures of connections, and recover from failure. These features are discussed in some detail below. Further details of the ADE system can be found in [19, 28, 3, 17, 18]; [19] in particular provides a comparison with many other agent development environments.

3.1. *Security Features*

ADE's security features rely in part on the Java security model.[b] All new ADE servers must be derived from the base `ADEServerImpl` abstract class, which encapsulates all of the ADE-related bookkeeping and credentials out of reach of the

[b]Of course, sophisticated users intent on bypassing the protections described here could implement a custom class loader to bypass, for example, `private` declarations. This is inherent to the Java security model, not specific to ADE's security features.

derived class by declaring these fields `private`. The superclass constructor makes the connection to the ADE registry, which requires a valid username and password to allow the server to register, thereby becoming part of the system.[c]

The ADE registry plays a key role in ensuring the security of the infrastructure. All methods in the `ADERegistryImpl` class are declared `final`, so users cannot bypass security methods by deriving a registry class and overwriting key methods. The registry checks to see, based on the server's credentials, whether it is allowed to join the system. At that point, the server is allocated a dynamically-generated key as part of the `heartbeat`, which is used both for security and fault-tolerance (see below). All communication between a server and the registry must be accompanied by that key, and verified against the registry's own key in the acknowledgement. Hence, even attempts to spoof `heartbeat` packets to access the registry will be difficult, as the correct (dynamically-generated) key must be provided.

The ADE registry has access to some important information in the `ADEServerImpl` via remote methods, which must be declared public. To protect the sensitive data, these methods require the calling class to pass as its credential an instance of an `ADERegistryImpl`, which the `ADEServerImpl` verifies using the `isInstanceOf` Java language primitive. Because the registry defines everything as `final`, only a class instance matching the already-defined registry class will be accepted as a valid credential.

When an ADE server needs a reference to another ADE server, it must contact the registry with the request, providing its credentials as described above. If the registry determines that the request is legitimate, it passes on the request to the remote server. Each server can set its own local access control for users, using usernames and passwords, to determine which other ADE servers should be allowed to use it. When the remote server accepts the request, subsequent interaction is peer-to-peer. A `heartbeat` is established between the two, including remote references to the server objects. These references are kept private by the `ADEServerImpl`, so a derived class does not have direct access to the remote resource. Instead, it is given a reference to the `heartbeat`, via which remote calls can be made and checked for access control by the `ADEServerImpl`. The servers exchange keys, ensuring that a new client cannot simply join and acquire a reference to another server.

These security measures ensure that, if architectural components are implemented in terms of ADE MAS agents, it will be possible to have very detailed control of access to functional components (e.g., the subset of servers with access to the parser component may be very different from the set given access to conceptual long-term memory). Hence, even though someone could inspect the system and look at the setup, that individual cannot examine internal states, request information,

[c]Users are prevented (by means of a `private` variable that can only be set by the `ADEServerImpl`) from bypassing these protections by instantiating an `ADEServerImpl`; instead, a derived class must call (in its own Java Virtual Machine) the `main` method of `ADEServerImpl`, which is `protected`, and therefore can only be called from a derived class. The `main` method then instantiates the derived class via reflection, which can then call the superclass constructor.

or make method calls without appropriate access rights. It is possible to build a system in which components are visible, yet malicious users cannot get references to them and thereby gain access to sensitive data. In an ambient intelligence environment, for example, service personnel may be allowed access to the system and to manipulate components relevant to their tasks without being able to gain access to personal settings, user preferences, and sensitive data.

3.2. *Reliability Features*

The ADE registry provides the backbone of an ADE system; all components must register to become part of the architecture. Hence, the registry is a key part of system reliability. For this reason, an ADE infrastructure may contain multiple concurrent ADE registries that mutually register with one another, providing both redundancy and the means of maintaining distributed knowledge about the system.

All connected components of an implemented architecture (i.e., ADE servers, ADE registries) maintain communication links during system operation, consisting of periodic heartbeat signals indicating that a component is still functioning. An ADE server sends a heartbeat to the registry with which it is registered. Similarly an ADE server also sends heartbeat messages to each ADE component to which it has a reference. The receiving component periodically confirms heartbeat reception; if none arrive, the sending component receives an error, while the receiving component times out. An ADE registry uses this information to determine the status of its servers, which in turn determines their accessibility. Similarly, an ADE server uses heartbeats to determine the status of its remote references, which determine if the remote server's services remain available.

Once a failure has been detected (via the heartbeat mechanism), the ADE infrastructure's *recovery* mechanisms can allow the system to restore itself to a fully running state. During the registration process, each ADE server provides information about how it was invoked, including command-line parameters, working directories, etc. The ADE registry uses this information to connect to the remote host (via a secure shell) and restart the Java process for that ADE server. In the case of a catastrophic host failure, the registry also has the option to restart the server on a different host. To support this functionality, each ADE server must inform the registry of what resources it requires to function correctly (e.g., a microphone for a speech recognition component) and each host must be defined in terms of the resources it makes available. When a match is available, the server is restarted on the new host. In some cases, it will not be possible to recover the failed server. When that happens, the ADE servers with references to that lost component are informed and can respond as appropriate (e.g., if the speech recognition component fails, the user could be informed: "There is a problem with my speech recognition, so you will need to use the keyboard interface to communicate with me."). More detail on, including experimental validation of, the reliability features described here can be found in [17].

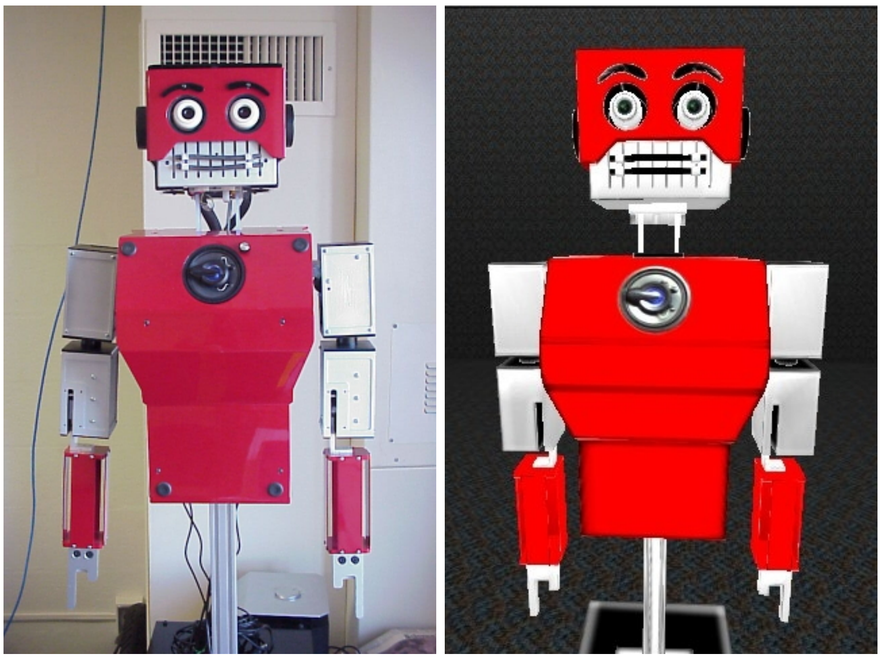

Fig. 1. A robotic platform for human-robot interaction (left) and its virtual representation in the USARSim simulation environment.

3.3. *Applications of ADE*

ADE has been developed primarily for the development of robotic control architectures. The system has been used with robots for several years in many experimental and demonstration scenarios. In this section, we provide an overview of many of the functional components that have been developed in ADE and are in current use in our lab. Subsequent sections examine in more detail specific components of particular interest.

The robot architecture DIARC (the "distributed integrated affect cognition and reflection" architecture) has been implemented in ADE, combining many of the features described here. DIARC is specifically designed for social robots that need to interact with humans in natural ways. It integrates cognitive capabilities (such as natural language understanding and complex action planning and sequencing) [32, 7, 6] with lower level activities (such as multi-modal perceptual processing, feature detection and tracking, and navigation and behavior coordination [33, 29]) and has been used in several human subject experiments and at various AAAI robot competitions [37, 36, 27, 26].

Multiple mobile robotic platforms are supported directly by ADE. Servers have been implemented that support the MobileRobots Pioneer P2-DX, P3-AT, and PeopleBot and the Segway RMP 200, in addition to custom platforms (e.g., a robotic golf cart). In addition, an ADE server is available that wraps the Player robot control system, making available a large number of additional platforms [10]. ADE also supports the RoboMotio Reddy robot torso (shown on the left of Figure 1). This robot, in concert with ADE servers for speech production and vision, provides a platform well-suited for human-robot interaction. In addition to verbal communication, the platform is capable of multi-modal non-verbal communication, including affect expression via facial expression (e.g., smiling, frowning), voice modulation (e.g., making the voice sound "stressed"), gesture (e.g., pointing to objects of interest), and shared-reference vision tracking (e.g., conveying understanding by turning to look at an object the user is describing). ADE also includes an implementation of the Reddy platform in the 3D simulation environment USARSim (Figure 1, right) that is presently being used to explore possible differences in how people respond to embodied versus virtual robotic agents. The virtual Reddy server implements the same interface as the physical Reddy, including facial expressions, gestures, and speech production. It can, therefore, be used as a drop-in replacement in exactly the same way as the real robot without requiring programming changes.

Despite the focus on robotic architectures, ADE development is not limited to the control of robots. In many cases it is beneficial to have supporting software packages integrated into the system (e.g., having all related components use ADE's built-in logging mechanisms can help with synchronization of timestamps, an important factor in subsequent analysis of experimental results). For example, an audio and video streaming architecture was created using ADE servers for recording and playing both media types; this streaming system allows events recorded in one area of the home to be displayed in another area, in real time. This functionality has been employed in experiments exploring the use of language in (human-human) exploration tasks [31]. Moreover, because the streaming architecture is implemented as a set of ADE servers, we are able to use it to aid in the distribution of multimedia analysis in a robot architecture (e.g., to stream onboard camera images to a remote machine for offboard processing).

Another example of a non-robotic application developed in ADE is a survey-administration system to assist in data collection for human-subjects experiments. The system allows users to use *action scripts* (see below) to specify questions for multiple response formats (e.g., text box, radio button, slider); the user's response is logged and returned to the script, allowing the experimenter to design "conditional" surveys (e.g., ignoring irrelevant lines of questioning based on user responses). This interface is currently being used to gauge subjects' impressions of robots in a (human-robot) exploration task, and would be useful in any situations in which verbal communication would be unwanted or awkward, as it is fully integrated into the ADE system.

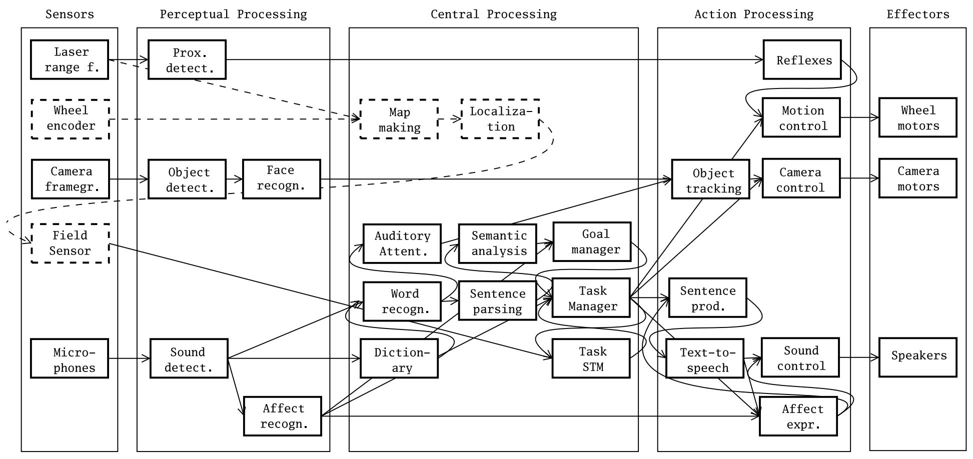

Fig. 2. A partial view of the DIARC robotic architecture for HRI consisting of only those components that were used in the experiment described here. Boxes depict concurrently running components of varying complexity and arrows indicate the information flow through the architecture. Dashed items are related to a simulated field sensor, and are not part of the architecture *per se* (see the experiment description).

4. Example Architecture

Figure 2 shows the subset of DIARC used in an experiment involving a hypothetical planetary exploration scenario (described briefly here; for further details, see [36]). The subject's task was to direct the robot in exploring an area, searching for a location from which the robot could transmit to an orbiting satellite. The subject could direct the robot's movements (e.g., "turn right," "go straight," etc.) and also ask for a reading of the current signal strength (simulated using an ADE "field" server that calculated the reported signal based on the robot's position in a map). When a location with a high enough signal strength was found, the subject would instruct the robot to transmit, and the experimental run was deemed a success. Performance was measured in terms of the time it took to locate the transmission point and transmit the data. Subjects were not told ahead of time that there was an artificial time limitation: after a fixed period of time, the robot would begin issuing low-battery reports, indicating the time remaining to complete the task. In some experimental conditions, the robot's voice was modulated to indicate affect (i.e., "fear" or "stress") after the first battery warning.

The flow of information in the architecture described here is depicted in Figure 3; this information flow is typical for ADE NLP configurations, although in many cases there are additional sensors/inputs and effectors/outputs. The speech recognizer provides words to the incremental parser (1), which can (2) instantiate a goal to (3) perform some actions during the parse (e.g., nodding to indicate agreement or

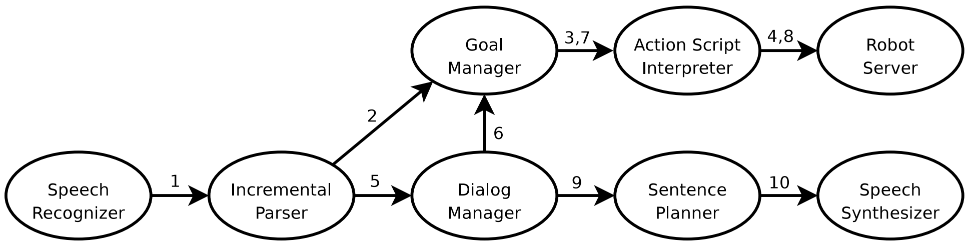

Fig. 3. NLP-related information flow in the example architecture. Ovals represent functional units of the architecture, and may correspond to more than a single box in Figure 2. Arrows indicate the (possibly parallel) flow of information, and the (temporally ordered) numbers on the arrows correspond to the events described in the text.

understanding while the speaker talks). When the parser has reached a conclusion, it (5) forwards the resulting semantic parse to the dialog manager. The dialog manager will sometimes (6) submit goals to the goal manager (e.g., when a command is recognized: “Turn left.”), and will often (9) initiate a verbal response in the sentence planner. The action script interpreter dispatches action requests as allowed by the goal manager, such as (4) sending a nod command to the robot server (or saying ‘OK, turning left.”) or beginning to execute the command (8). The sentence planner then passes the generated sentences to the speech synthesizer (10) for output.

Analysis of the results indicated that affect expression on the part of the robot improved subjects’ performance ([36]), both in terms of time to complete the task and in terms of success rate, over subjects in the control condition. Expressing affect seems to induce a sense of urgency in humans, even when coming from a very mechanistic, non-humanlike robot, the PeopleBot. This is just one example of successful deployment of an architecture constructed using ADE; for others, see the references.

4.1. *Goal Management*

The DIARC goal manager’s job is to determine which actions should be performed at any given time. It does this by assessing the priority of each goal associated with an action sequence that can achieve it (e.g., the housekeeper’s goals of keeping the building clean and remaining unobtrusive) and awarding resources to high-priority goals. A goal’s priority is based on its *expected utility* and its *urgency*. The expected utility $u = p \cdot b - c$, where p is an estimate of the probability of achieving the goal (e.g., based on prior experience), b is the benefit of achieving the goal, and c is the cost of attempting to achieve the goal. The urgency is based on the time remaining within which to achieve the goal: $g = \frac{t_e}{t_a} \cdot (g_{max} - g_{min}) + g_{min}$, where t_e and t_a are the time elapsed so far in pursuing the goal and the total time allowed to pursue the goal, and g_{max} and g_{min} are upper and lower bounds on the urgency of that particular goal. Hence, as the time spent working towards a goal approaches the maximum time allowable for that goal, the urgency rises, subject to upper and lower bounds. The goal's priority $p = u \cdot g$.[d]

The goal manager uses the priorities to control which goals have access to which resources. When there is no conflict over resources, multiple goals can be serviced concurrently. For example, when there is no human in the building, the two housekeeper goals (*unobtrusiveness* and *cleaning*) are not in conflict, so it can clean. In general, when there is a human present, the higher-priority unobtrusiveness goal gets control of the robot's motors so that the cleaning goal cannot operate. However, if that priority is overridden (e.g., by a human calling to have a spill cleaned up), the cleaning goal has higher priority and is able to assume control.

The knowledge base includes factual knowledge relevant to the goals of the agent (e.g., the preferences of the human with regard to drinking scotch with dinner, when the human typically arrives home at the end of the day, visual cues used to distinguish individual humans) as well as procedural knowledge of how to accomplish goals and subgoals. Procedural knowledge is encoded in the form of *action scripts*, specifications of the steps required to perform a task, along with information about the other agents, people, and objects that are relevant to the task.

4.2. *Natural Language Processing*

Figure 4 gives a NLP-centric view of DIARC, showing how the discourse manager fits into the architecture while providing a more detailed view of the discourse manager. Information flows are indicated by the arrows. The ADE speech recognition and vision servers process input from hardware sensors, generating representations of the data to be used by TIDE, the *Timing-sensitive Incremental Discourse Engine* (described briefly below, and in more detail in [5, 6]). Sound data is converted to words by the speech recognition server, and visual data into a situation model represented in visual short term memory (VSTM) and maintained by the vision server. In the course of processing an utterance, TIDE relies on the vision server for information about the visual environment and interacts with the action server to provide back-channel feedback and head movements for referent confirmation. Verbal responses are sent to the speech production server, and any commands that TIDE interprets are passed to the goal management system, where they are evaluated for execution as described above.

Natural language understanding is handled by the *robotic incremental semantic engine* (RISE) [7], part of an ADE discourse server, which incrementally processes utterances as they are provided by the speech recognizer (typically word-by-word). Incremental processing allows the integration of perceptions (e.g., information from the vision subsystem that helps with reference resolution). As new words arrive, RISE reduces the size of the set of possible referents based on the words' associated syntactic constraints and semantic constraints established after meaning is

[d]For further details on ADE's goal management component, see the references, e.g., [35]

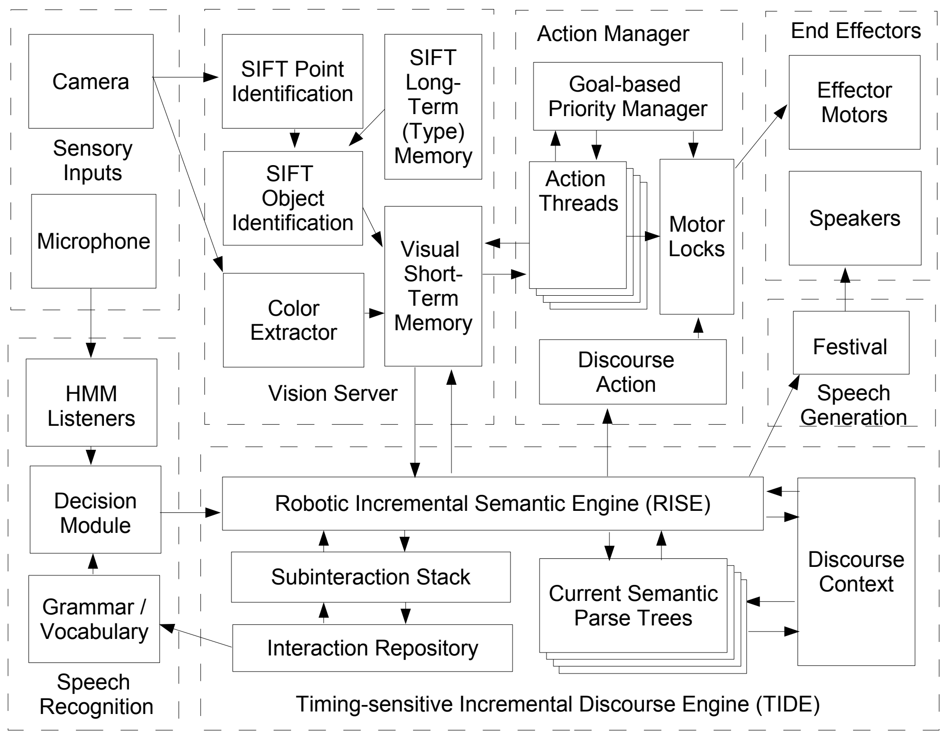

Fig. 4. A representation of the DIARC architecture from the perspective of TIDE, ADE's dialog management component.

established. For example, given the command "put the glass on the floor in the refrigerator" in a scenario in which there is a glass on the table and a glass on the floor, RISE is able to correctly identify the referent once the "on the floor" has been processed. This allows more natural interactions and avoids problems such as the overspecification of referents, which can cause confusion [8]. Moreover, processing incrementally allows RISE to suggest actions (e.g., demonstrating understanding by looking at the glass on the floor once the reference has been resolved) that can facilitate smooth interactions.

In addition, incremental processing helps make the system more robust to misunderstood or incomplete speech. Discourse management in ADE is done by TIDE. TIDE provides a *discourse context* structure that can be used in the resolution of anaphoric expressions in conversation and an *interaction template* structure that contains semantic and syntactic structures for words and their associated meanings in the context of commonly-seen interactions/sentences. The discourse context keeps an annotated stack of past referents. When an anaphoric expression needs to be resolved, for example, following up the previous command with "no, just dump it in the sink," there "it" refers to the glass on the floor, TIDE examines the stack and selects the most likely recent referent that meets syntactic and semantic guidelines.

TIDE's interaction templates provide information about which syntactic/semantic elements should be present in a given sentence. As a sentence is interpreted, these elements are retrieved from the interaction template and used in a constraint-propagation system to incrementally determine the meaning of an utterance. In the event that a necessary binding cannot be found, a subinteraction is added to the interaction stack and a clarifying question or set of questions is generated. The subinteraction adds the context of a limited interaction meant to resolve the missing variable (i.e., referent) bindings. For example, if the user issues the command "change to disc 3" but the number is somehow garbled, TIDE can determine from the template that it needs a number before it can create a complete command request, and it generates a question to clarify ("What number should I change to?"). This approach to natural language understanding allows for much more natural interactions between users and the system.

5. Discussion

ADE's combination of distributed computation, security, fault tolerance, and functionality make it a good fit for ambient intelligence applications, such as a smart home. While no particular smart home architecture has been implemented in ADE yet, many of the already deployed ADE systems used components (i.e., ADE servers) that could be used in smart home applications. Because of ADE's focus on mobile robots, many of the challenges that are posed by ambient intelligence are already solved, as they are required to effectively control robotic platforms. Specifically, robots rely heavily on multiple sensor modalities (e.g., laser rangefinders, cameras, and microphones) that require varying degrees of cognitive complexity to translate into useful information, from simple reactive "broken beam" detection for obstacle avoidance to complex visual analysis of images in real time. Moreover, natural language processing plays a crucial role in human-robot interaction, requiring fast, responsive semantic analysis of speech input.

ADE's NLP subsystem has been qualitatively validated in multiple experimental setups demonstrating aspects of the system such as the use of visual perceptions to aid resolution of disfluencies [7] to the integration of incremental language processing with action management [6]. However, we have yet to embark on a quantitative evaluation of the system's performance. That said, issues such as system scalability are addressed in ADE's design; given the complexity of NLP and other tasks in both human-robot interaction and ambient intelligence, effective system control will often require the resources of multiple computers that must communicate via networks. ADE has been successfully deployed in multi-robot scenarios, and SWAGES [34], a version of ADE, is routinely used for distributed high-performance computing for agent-based modeling on hundreds of hosts.

In this section, we present an example architecture for ambient intelligence for

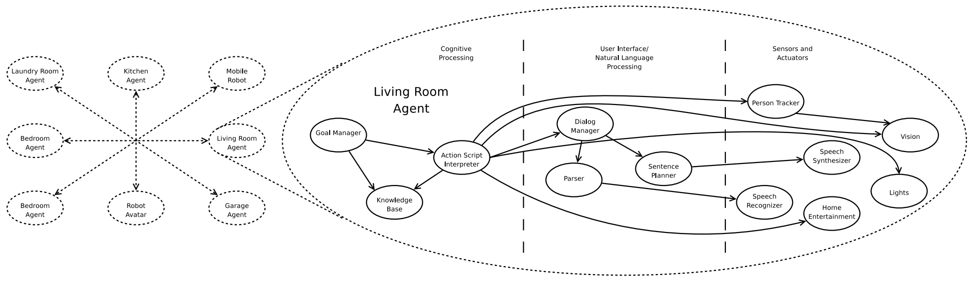

Fig. 5. Overview of a subset of the smart house architecture. Dotted ovals represent SAS ambient intelligence agents. Dotted arrows indicate logical connections between SAS agents. Solid ovals represent ADE MAS agents, including goal management, NLP, and servers for sensors and actuators. Solid arrows indicate references in the MAS agents from which they originate to the MAS agents to which they point.

the scenario of a "smart building" (e.g., a home, as described in the introduction). This example is intended to demonstrate some of the types of features made available by ADE, not to advocate a particular design over any other.[e] The example system combines multiple sensors, effectors, and software agents of varying complexity to allow users to perform routine tasks via a natural language interface.

The example presented here is an architecture that allows users to:

- request information from the system (e.g., sensor readings)
- operate simple devices (e.g., changing the channel on the television)
- monitor and log sensor readings and state changes within the system (e.g., to monitor the activity of a resident to facilitate independent living)
- control mobile robots for household tasks

Figure 5 shows a subset of the "smart home" architecture. It depicts a number of high-level SAS agents and zooms in on the `living room agent` to depict its constituent ADE MAS agents. The figure is simplified for the sake of clarity; in a realistic rendering of the architecture, there may be more SAS (room) agents, and each of the agents would have references to many more ADE servers. Notice that the SAS agents can have references to each other directly, without having to send requests to a (yet) higher-level house agent (again, the details of how they determine when to communicate with another agent and how they determine which ones to contact are beyond the scope of this paper). Several ADE servers will be instantiated multiple times with different configurations, associated with different SAS agents (e.g., NLP, Vision). This allows for specialization of the server based on the scenario, such as limiting the grammar and vocabulary of the NLP component or tuning the vision component to conditions in a particular room. Many SAS agents

[e]While the system as described here has not been implemented/realized in this specific configuration, all individual components have been implemented and used in a variety of different scenarios.

can hold references to a single ADE server (e.g., the person tracker). In this case, the room agents have references to the person tracker and to their individual vision servers, in addition to the person tracker having references to the individual rooms' vision servers.

The architecture for this scenario includes a "base" room package (e.g., the Living Room Agent in Figure 5, but without references to specialized ADE servers such as the home entertainment server) that is available in each room of the building, including a set of common sensors (e.g., light, motion, temperature, microphone) and actuators (e.g., light switch, thermostat, window shades, speakers). The base agent includes a very simple *goal manager*, a limited *knowledge base*, and an *action script interpreter* (more detailed descriptions of these components can be found in [35, 20]).

A library of action scripts is needed to handle the variety of requests that may be made by humans or initiated by the reasoning components of the smart home. The action script interpreter routes calls for sensing or actuating events to the appropriate ADE servers. So, for example, when the `living room agent` is informed that you are on your way down the hall, it starts an action script that instructs the light switch server to bring up the lights, makes a knowledge base query regarding your TV show preference, and instructs the home entertainment server to turn on the TV and tune to that show:

```
script: prepareRoom

agent: livingroom
agent: person
name: show

activateLights(livingroom)
show := queryPrefs(livingroom, person, "TV show", currenttime)
activateTV(livingroom)
selectTVshow(livingroom, show)
```

For the simple room agent, the goal manager has a single goal: to listen for and carry out user requests (e.g., for information, or actions to be performed). Given this simple scenario, the natural language processing capabilities of the base room are fairly simple. Some scenarios require specialized goal management and capabilities, tailored to fit the needs and desires of the user in those scenarios. For example, in a living or family room, the user may want to be able to control an entertainment system. This requires the system to understand many queries (e.g., "What is the title of this track?" "How much time is remaining?" "Are there other songs by this artist?"), as well as many commands (e.g., "Turn up the volume," "Change to disc 3," "Switch to the satellite receiver and turn on the television."). The goal manager is provided with scripts corresponding to each of these, and can supervise the execution of multiple scripts in parallel, unless there are conflicts (e.g., a request that would require playing two discs at once). The natural language processing demands on this agent are substantial, given the potential variety of (song, show, movie, etc.) titles. This is a case in which incremental language processing with contextual feedback is highly beneficial; the system can have access to the library of CDs, the shows being broadcast currently (or in the near future), etc., and can use that information to help rule out possibilities. For example, suppose you tell the system, "Play the first Star Trek movie—no, actually, play the Cure." In this case, "'the Cure" is both a movie and a band, but given the context (i.e., your initial request for a movie) the system is able to select the more likely one and start the movie. The base package can also easily be extended to include cameras and visual processing capabilities that would be useful in providing semantic constraints during NLP (e.g., disambiguation between similar objects based on color).

These examples are all contained within a single room. However, there are many ways to extend the architecture further. For example, the smart home could track multiple people concurrently, allowing easy access to information ("where are the kids?") and communication ("let the kids know it's time for dinner"). The information could be used in many ways, such as user-configurable controls (e.g., restrictions on which devices can be operated by whom, or limits on their operation) and alarms (e.g., to inform an adult when a child enters a potentially dangerous area). Moreover, ADE's distribution capabilities allow the rooms to be connected to form a coherent system that extends the capabilities of the "room agent" for any particular room to include many of the capabilities in more complex agents from other scenarios. Should the agent in the current room fail to understand the speaker, however, the system is able to pass on the request to another (more complex) agent, at the potential cost of increased response time. For example, a user in the bedroom could give the command "Heat up the pasta from last night." The bedroom agent's NLP server does not include food preparation vocabulary, so it does not understand the request. It can "broadcast" the request to other agents to find one that can handle it; ADE servers for audio streaming relay the user's speech to the remote servers. When the `kitchen agent` recognizes the command, it can begin warming up the leftovers and send the response ("The pasta will be ready in five minutes.") back to the bedroom. Hence, the underlying system is actually composed of multiple independent agents; an open question is whether people will prefer a system in which multiple agents are present (e.g., with each room agent having a different representation, such as an individualized avatar), or one in which it appears to the user as though there is a single "house" agent, consistent throughout the home. With ADE, either is possible, and it is a matter of determining what a particular individual prefers.

ADE's support for multiple mobile robot platforms will be useful in a smart building scenario. Although currently the technology does not exist for all the smart devices described above (actuators to load the microwave with food from the refrigerator, etc.), there are robots that are designed for the purpose of interacting with people and may be good additions to the smart home. For example, some users may not like speaking to a "disembodied voice" and would prefer a visual representation of the various room agents. The use of a mobile robot like this allows more sophisticated monitoring in addition to the base capabilities of the room agents. For example, the robot platform can be equipped with more sensitive or expensive sensors than those placed in each room. The robot then serves as a mobile sensing platform, moving from room to room, as needs demand, providing extended coverage with the sophisticated sensors without multiplying their cost. Moreover, the robot has access to all of the information in the system (via the other agents), allowing it to efficiently perform its tasks without requiring the user to reiterate preferences, desires, etc., as would be required by a robot that was not integrated into the home system.

Similarly, the simulated version of the Reddy robot can be used as a visual interface to the architecture. Hence, it can serve as the "face" of the ambient intelligence system in supplement to the room agents throughout the building. ADE's distribution mechanisms make it easy to display the avatar for the `kitchen agent` on a screen in the living room, for example, to check whether you want your dinner delayed. This makes it possible to give people the impression that the robot is in charge of the house's operation, although, as stated above, it is probably a matter of preference which people will like better, a single agent, or multiple agents.

As described above, ADE's security features are designed to prohibit unauthorized access to ADE servers. This prevents malicious access to sensitive information (e.g., streaming audio, personal preferences, etc.) via the wireless network. Moreover, ADE has built-in mechanisms for fault tolerance and recovery to help prevent system failures. For example, although the architecture can be organized hierarchically, even if a higher-level system or agent should fail, the lower (room) level agents would continue to operate as normal, modulo the ability to pass on requests outside the room. In the meantime, ADE's recovery mechanisms will initiate restarting the affected servers, bringing the system up to full functionality without interfering with local functionality, barring any unrecoverable hardware failure. If the problem should be due to a hardware failure, ADE has the ability to select other hardware with similar capabilities, if available, on which to restart the failed server. We have conducted experiments to verify that the recovery mechanisms can operate "behind the scenes," restoring ADE servers without users noticing they had failed [20].

6. Conclusion

Ambient intelligence systems pose many challenges to developers, ranging from interfacing with a large number of distinct devices to communication in a complex network environment. In fact, those challenges are not unique: robot architectures, particularly complex cognitive architectures for human-robot interaction, must often solve many of the same problems. The ADE infrastructure addresses the security, communication, and reliability issues necessary for ambient intelligence, and provides many interfaces to sensors that would be of useful in ambient semantic computing systems. One major challenge for any such system is fast, accurate natural language processing. ADE's incremental natural language processing capabilities address both issues, increasing speed performance by starting processing before the utterance is complete and by allowing the system to discard incorrect possibilities earlier than would otherwise be possible. Other sensing modalities, such as vision processing, provide additional information that can be used to further refine parsing and overall NLP performance [32].

ADE's support for robotic platforms allows mobile agents to be tightly integrated into the smart home architecture; mobile robots will be very useful for the automation of many simple (cleaning, delivery) tasks, as well as allowing for fine-grained dynamic control of sensor placement (e.g., for elder care monitoring). In addition, virtual implementations of robot platforms provide natural alternatives for visual agent representations (avatars) where video monitors are present. This makes it possible to remove the physical robot from the system entirely, if desired, while still providing a "personal" interface to the smart home. The virtual domain is also very useful for prototyping and testing purposes; it is likely that the optimal configuration of a smart building will be different for different people, and the virtual environment and agents allow for easy experimentation with features to find the best setup for a particular individual.

ADE also has sophisticated mechanisms for distributed computing, allowing systems to share resources remotely. The ability to operate multiple goal managers concurrently with differing degrees of cognitive complexity makes it possible to tailor the individual agents (e.g., the "room agent") to the scenario; this can also be very beneficial to NLP, as the language domain can be simplified to speed up semantic analysis. Moreover, ADE's distributed SAS/MAS agent model makes reconfiguring the system simple. For example, to change a room's configuration, or add another agent for a new area, it is not necessary to write new code and recompile the whole software package. Instead, the startup configuration can be changed to specify a new set of ADE servers and their connections (to each other). Selecting a different language model for a particular room is as simple as specifying a different configuration file for the NLP server. Finally, new servers can be brought online at runtime as well, allowing dynamic reconfiguration of the system.

In short, ADE is a distributed agent infrastructure that has natural language processing built in and is connected to sophisticated goal management mechanisms which interact with the world via multiple server interfaces for sensing and actuating devices. We are currently working with HRI experimental setups that contain many of the features of the described smart home environment, and are planning to implement some of the findings in user studies in a smart home environment for real-world tests. Servers for more sensors and devices relevant to general ambient intelligence systems are being implemented, and we are working on improving the incremental natural language processing capabilities of the system to increase speed and accuracy via semantic constraints. Finally, testing in the virtual environment needs to be conducted, to determine whether and how people respond differently to virtual agents (such as room agent avatars) than to physically present agents (such as an autonomous mobile robot).

References

[1] Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. MundoCore: A Lightweight Infrastructure for Pervasive Computing. *Pervasive and Mobile Computing*, 2007.

[2] Virgil Andronache and Matthias Scheutz. Integrating theory and practice: The agent architecture framework APOC and its development environment ADE. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, pages 1014–1021, 2004.

[3] Virgil Andronache and Matthias Scheutz. ADE—an architecture development environment for virtual and robotic agents. *International Journal of Artificial Intelligence Tools*, 15(2):251–286, 2006.

[4] Christian Becker, Gregor Schiele, Holger Gubbels, and Kurt Rothermel. Base—a micro-broker-based middleware for pervasive computing. *IEEE International Conference on Pervasive Computing and Communications*, 0:443, 2003.

[5] Timothy Brick. TIDE: A timing-sensitive incremental discourse engine. Master's thesis, University of Notre Dame, Notre Dame, IN, 2007.

[6] Timothy Brick, Paul Schermerhorn, and Matthias Scheutz. Speech and action: Integration of action and language for mobile robots. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1423–1428, San Diego, CA, October/November 2007.

[7] Timothy Brick and Matthias Scheutz. Incremental natural language processing for HRI. In *Proceedings of the Second ACM IEEE International Conference on Human-Robot Interaction*, pages 263–270, Washington D.C., March 2007.

[8] Kathleen Eberhard, Michael Spivey-Knowlton, Julie Sedivy, and Michael Tanenhaus. Eye movements as a window into real-time spoken language comprehension in natural contexts. *Journal of Psycholinguistic Research*, 24:409–436, 1995.

[9] FIPA agent management specification (SC00023K). http://www.fipa.org/specs/fipa00023/, 2004.

[10] Brian Gerkey, Richard Vaughan, and Andrew Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, June 2003.

[11] Robert Grimm. One.world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, 3(3):22–30, 2004.

[12] IST Advisory Group. Ambient intelligence: from vision to reality. In Giuseppe Riva, Francesco Vatalaro, Fabrizio Davide, and Mariano Alcaniz, editors, *Ambient Intelligence.* IOS Press, 2005.

[13] Gerd Herzog, Alassane Ndiaye, Stefan Merten, Heinz Kirchmann, Tilman Becker, and Peter Poller. Large-scale software integration for spoken language and multimodal dialog systems. *Natural Language Engineering*, 10:283–305, 2004.

[14] Gerd Herzog and Norbert Reithinger. The SmartKom architecture: A framework for multimodal dialogue systems. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems.* Springer-Verlag, Secaucus, NJ, 2006.

[15] Nicholas Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[16] Anthony Joseph, Joshua Tauber, and M. Frans Kaashoek. Mobile computing with

the Rover toolkit. *IEEE Transactions on Computers*, 46(3):337–352, 1997.

[17] James Kramer and Matthias Scheutz. ADE: A framework for robust complex robotic architectures. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4576–4581, Bejing, China, October 2006.

[18] James Kramer and Matthias Scheutz. ADE: Filling a gap between single and multiple agent systems. In *Proceedings of the ACE 2004 Symposium at the 18th European Meeting on Cybernetics and Systems Research*, Vienna, Austria, 2006.

[19] James Kramer and Matthias Scheutz. Robotic development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[20] James Kramer, Matthias Scheutz, and Paul Schermerhorn. 'Talk to me!': Enabling communication between robotic architectures and their implementing infrastructures. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3044–3049, San Diego, CA, October/November 2007.

[21] John Laird, Allen Newell, and Paul Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[22] David Leake, editor. *Case-Based Reasoning: Experiences, Lessons, and Future Directions.* MIT Press, Cambridge, MA, 1996.

[23] Bruce A. Maxwell, William Smart, Adam Jacoff, Jennifer Casper, Brian Weiss, Jean Scholtz, Holly Yanco, Mark Micire, Ashley Stroupe, Dan Stormont, and Tom Lauwers. 2003 AAAI robot competition and exhibition. *AI Magazine*, 25(2):68–80, 2004.

[24] Thomas Portele, Silke Goronzy, Martin Emele, Andreas Kellner, Sunna Torge, and Jürgen te Vrugt. SmartKom-Home: The interface to home entertainment. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems.* Springer-Verlag, Secaucus, NJ, 2006.

[25] Alessandro Saffiotti and Mathias Broxvall. Peis ecologies: ambient intelligence meets autonomous robotics. In *SOC-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 277–281, New York, NY, USA, 2005. ACM.

[26] Paul Schermerhorn, James Kramer, Timothy Brick, David Anderson, Aaron Dingler, and Matthias Scheutz. DIARC: A testbed for natural human-robot interactions. In *Proceedings of AAAI 2006 Robot Workshop*, 2006.

[27] Paul Schermerhorn, Matthias Scheutz, and Charles R. Crowell. Robot social presence and gender: Do females view robots differently than males? In *Proceedings of the Third ACM IEEE International Conference on Human-Robot Interaction*, Amsterdam, NL, March 2008.

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

[29] Matthias Scheutz and Virgil Andronache. Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. *IEEE Transactions of System, Man, and Cybernetics Part B*, 34(6):2377–2395, 2004.

[30] Matthias Scheutz, Virgil Andronache, James Kramer, Philip Snowberger, and Eric Albert. Rudy: A robotic waiter with personality. In *Proceedings of AAAI Robot Workshop.* AAAI Press, 2004.

[31] Matthias Scheutz and Kathleen Eberhard. Towards a framework for integrated natural language processing architectures for social robots. In *Proceedings of the 5th International Workshop on Natural Language Processing and Cognitive Science*, pages 165–174, Barcelona, Spain, June 2008.

[32] Matthias Scheutz, Kathleen Eberhard, and Virgil Andronache. A parallel, distributed, realtime, robotic model for human reference resolution with visual constraints. *Con-*

nection Science, 16(3):145–167, 2004.

[33] Matthias Scheutz, John McRaven, and Gyorgy Cserey. Fast, reliable, adaptive, bimodal people tracking for indoor environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1340–1352, 2004.

[34] Matthias Scheutz, Paul Schermerhorn, Ryan Connaughton, and Aaron Dingler. SWAGES–an extendable parallel grid experimentation system for large-scale agent-based alife simulations. In *Proceedings of Artificial Life X*, pages 412–418, June 2006.

[35] Matthias Scheutz, Paul Schermerhorn, James Kramer, and David Anderson. First steps toward natural human-like HRI. *Autonomous Robots*, 22(4):411–423, May 2007.

[36] Matthias Scheutz, Paul Schermerhorn, James Kramer, and Christopher Middendorff. The utility of affect expression in natural language interactions in joint human-robot tasks. In *Proceedings of the 1st ACM International Conference on Human-Robot Interaction*, pages 226–233, 2006.

[37] Matthias Scheutz, Paul Schermerhorn, Christopher Middendorff, James Kramer, Dave Anderson, and Aaron Dingler. Toward affective cognitive robots for human-robot interaction. In *AAAI 2005 Robot Workshop*, pages 1737–1738. AAAI Press, 2005.