AI Magazine Volume 25 Number 4 (2004) (© AAAI)

Constructionist Design Methodology for Interactive Intelligences

Kristinn R. Thórisson, Hrvoje Benko, Denis Abramov, Andrew Arnold, Sameer Maskey, and Aruchunan Vaseekaran

■ We present a methodology for designing and implementing interactive intelligences. The *constructionist design methodology* (CDM)—so called because it advocates modular building blocks and incorporation of prior work—addresses factors that we see as key to future advances in AI, including support for interdisciplinary collaboration, coordination of teams, and large-scale systems integration.

We test the methodology by building an interactive multifunctional system with a real-time perception-action loop. The system, whose construction relied entirely on the methodology, consists of an embodied virtual agent that can perceive both real and virtual objects in an augmented-reality room and interact with a user through coordinated gestures and speech. Wireless tracking technologies give the agent awareness of the environment and the user's speech and communicative acts. User and agent can communicate about things in the environment, their placement, and their function, as well as about more abstract topics, such as current news, through situated multimodal dialogue.

The results demonstrate the CDM's strength in simplifying the modeling of complex, multifunctional systems that require architectural experimentation and exploration of unclear subsystem boundaries, undefined variables, and tangled data flow and control hierarchies.

The creation of emboïdied humanoids and broad AI systems requires integration of a large number of functions that must be carefully coordinated to achieve coherent system behavior. We are working on formalizing a methodology that can help in this process. The architectural foundation we have chosen for the approach is based on the concept of a network of interacting modules that communicate via messages. To test the design methodology we chose a system with a human user that interacts in real time with a simulated human in an augmented-reality environment. In this article we present the design methodology and describe the system that we built to test it.

Newell (1990) urged for the search of unified theories of cognition,[1] and recent work in AI has increasingly focused on integration of multiple systems (compare with Simmons et al. 2003, McCarthy et al. 2002, and Bischoff and Graefe 1999). Unified theories necessarily mean integration of many functions, but our prior experience in building systems that integrate multiple features from artificial intelligence and computer graphics (Bryson and Thórisson 2000, Lucente 2000, Thórisson 1999) has made it very clear that such integration can be a challenge, even for a team of experienced developers. In addition to basic technical issues—connecting everything together can be prohibitive in terms of time—it can be difficult to get people with different backgrounds, such as computer graphics, hardware, and artificial intelligence, to communicate effectively. Coordinating such an effort can thus be a management task of a tall order; keeping all parties synchronized takes skill and time. On top of this comes the challenge of deciding the scope of the system: what seems simple to a computer graphics expert may in fact be a long-standing dream of the AI person, and vice versa.

Several factors motivate our work. First, a much-needed move towards building on prior work in AI to promote incremental accumulation of knowledge in creating intelligent systems is long overdue. The relatively small group working on broad models of mind, the

researchers who are bridging across disciplines, need better ways to share results and work together and to work with others outside their fields. To this end our principles foster reusable software components through a common middleware specification[2] and mechanisms for defining interfaces between components. Second, by focusing on the reuse of existing work, we are able to support the construction of more powerful systems than otherwise possible, speeding up the path towards useful, deployable systems. Third, we believe that to study mental mechanisms they need to be embedded in a larger cognitive model with significant breadth to contextualize their operation and enable their testing under boundary conditions. This calls for an increased focus on supporting large-scale integration and experimentation. Fourth, bridging across multiple functions in a single, unified system increases researchers' familiarity and breadth of experience with the various models of thought to date—as well as new ones. This is important—as are in fact all of the above points—when the goal is to develop unified theories of cognition.

Inspired to a degree by the classic LEGO bricks, our methodology—which we call a *constructionist approach to AI*—puts modularity at its center: Functions of the system are broken into individual software modules, which are typically larger than software classes (that is, objects and methods) in object-oriented programming but smaller than the typical enterprise application. The role of each module is determined in part by specifying the message types and content of the information that needs to flow between the various functional parts of the system. Using this functional outline we then define and develop, or select, components for perception, knowledge representation, planning, animation, and other desired functions.

Behind this work lies the conjecture that the mind can be modeled through the adequate combination of interacting functional machines (modules). Of course, this is still debated in the research community, and not all researchers are convinced of the idea's merits. However, this claim is in its essence simply a combination of two less radical ones. First, that a divide-and-conquer methodology will be fruitful in studying the mind as a system. Since practically all scientific results since the Greek philosophers are based on this, it is hard to argue against it. In contrast to the search for unified theories in physics, we see the search for unified theories of cognition in the same way as articulated in Minsky's (1986) theory, that the mind is a multitude of interacting components, and his (perhaps whimsical but fundamental) claim that the brain is a hack.[3] In other words, we expect a working model of the mind to incorporate, and coherently address, what at first seems a tangle of control hierarchies and data paths. Which relates to another important theoretical stance: The need to model more than a single or a handful of the mind's mechanisms in isolation in order to understand the working mind. In a system of many modules with rich interaction, only a model incorporating a rich spectrum of (animal or human) mental functioning will give us a correct picture of the broad principles underlying intelligence.

There is essentially nothing in the constructionist approach to AI that lends it more naturally to behavior-based AI (compare with Brooks 1991) or "classical" AI—its principles sit beside both. In fact, since constructionist AI is intended to address the integration problem of very broad cognitive systems, it must be able to encompass all variants and approaches to date. We think it unlikely that any of the principles we present will be found objectionable, or even completely novel for that matter, by a seasoned software engineer. But these principles are custom-tailored to guide the construction of large cognitive systems, and we hope it will be used, extended, and improved by many others over time.

To test the power of a new methodology, a novel problem is preferred over one that has a known solution. The system we chose to develop presented us with a unique scope and unsolved integration issues: an augmented reality setting inhabited by an embodied virtual character; the character would be visible via a see-through stereoscopic display that the user wears and would help the user navigate the real-world environment. The character, called Mirage, should appear as a transparent, ghostlike stereoscopic 3-D graphic superimposed on the user's real-world view (figure 1). This system served as a test bed for our methodology; the system is presented in sufficient detail here to demonstrate the application of the methodology and to show the methodology's modular philosophy, which the system mirrors closely.

Related Work

Related work can be broadly categorized into three major sections based on emphasis: modularization, mind models, and embodiment. Our work on constructionist AI directly addresses the first two while embodiment is more relevant to the project that we used to test the methodology.

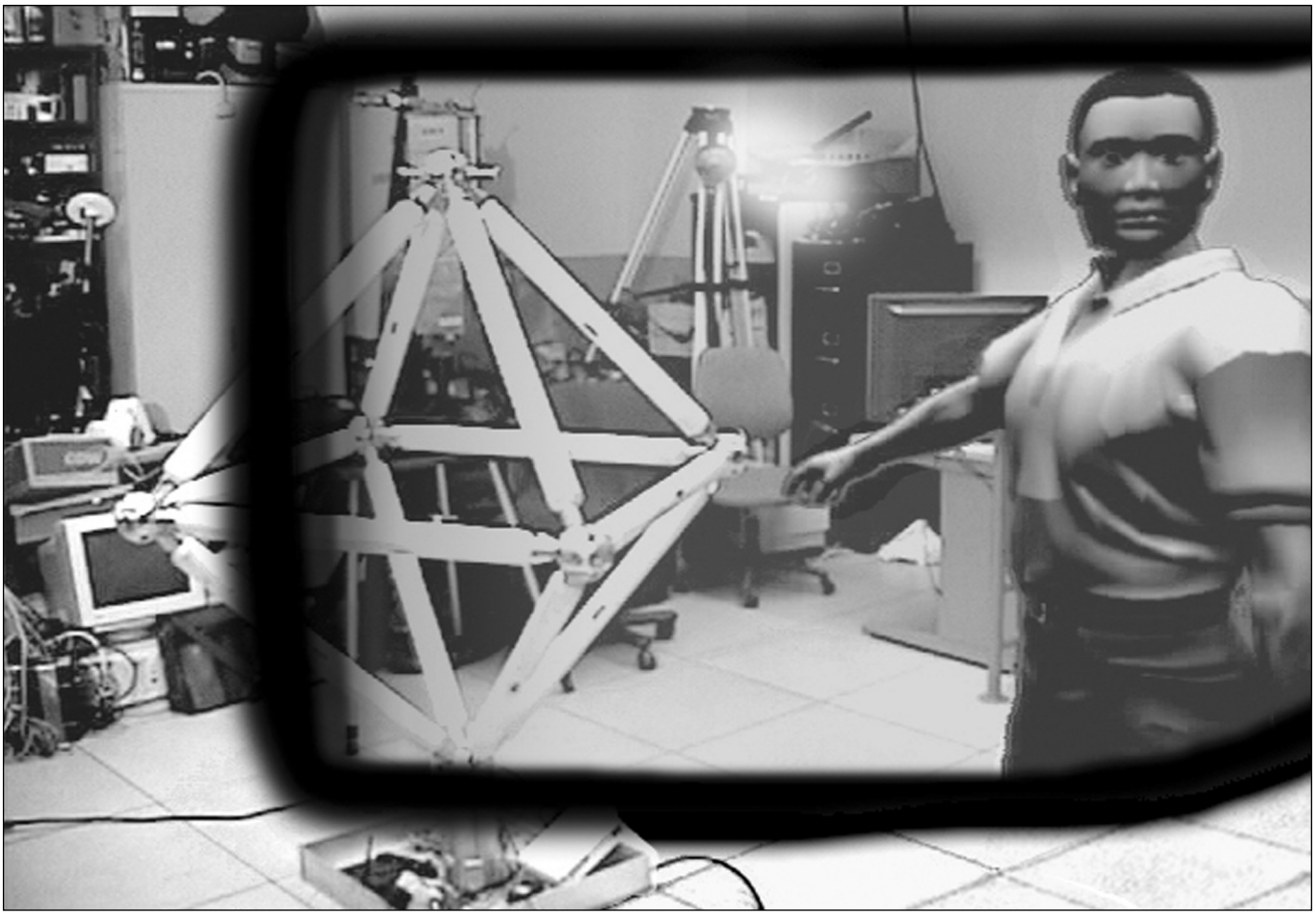


Figure 1. Our Embodied Agent Mirage Is Situated in the Laboratory.

Here we see how Mirage appears to the user through the head-mounted glasses. (The image has been enhanced for clarity.)

Modularity

Modular approaches have been shown to speed up the development of large, complex systems and to facilitate the collaborations of large teams of researchers (Fink et al. 1995, 1996). For instance, Martinho, Paiva, and Gomes (2000) describe an architecture designed to facilitate modular, rapid development of theatrical agents in virtual worlds. Like many other systems (compare with Granström, House, and Karlsson 2002; Laird 2002; McKevitt, Ó Nulláin, and Mulvihill 2002; Badler, Palmer, and Bindiganavale 1999; and Terzopolous 1999), their agents live in an enclosed virtual world where the update cycle of the world can be artificially controlled. Martinho, Paiva, and Gomes's architecture involves splitting the world, along with the agents, into three separate modules, each of which can be developed independently of the others. As in our approach, this enables parallel execution of implementation, which reduces development time, and combining the pieces to form a full system becomes a simpler task. Modularity also played a large role in the construction of Bischoff and Graefe's (2000) impressive Hermes robot. Simmons and colleagues' robot Grace (2003), which involved the work of five institutions and more than 20 people, is another fine example of a project that has benefited from a modular approach.

The emphasis on modularity in our constructionist methodology derives primarily from the approach of Thórisson (1996), first used for the communicative humanoid Gandalf and later for work at LEGO Digital (Bryson and Thórisson 2000), and from behavior-oriented design, or BOD (Bryson 2002). Bryson details the elements of BOD, using principles from behavior-based AI, such as reactive plans and tight integration of perception and action. However, this approach encompasses principles focused on particular ways of planning and lacks more general principles and ways of using preexisting modules. The Ymir architecture (Thórisson 1999) presents relatively gener-

al modularization principles, using module types such as perceptors, deciders, and actions with subtypes, in a way that clearly distinguishes their roles and mechanisms. This was formalized and incorporated into the present work. We further extend it by turning the conventional module-oriented build process on its head by making the messages in the system equally as important as the modules themselves, as explained further below.

Embodiment

In bringing together an agent and a user, two methods are most common: the agent is brought into the user's world, as in robotics or screen-based agents,[4] or the user is brought into the agent's world, as in immersivevirtual world simulations.

Due to their physical presence in the real world, robots are inherently embodied. There has been a recent surge in simulating humanlike movement and decision-making processes in robotics, and even more recent have been attempts to create a goal-oriented humanlike interaction between the robot and the user. An example is the work of Iba, Paredis, and Khosla (2002), where multimodal input gets used (gestures and speech) to program the vacuum-cleaning robot. Another version of bringing the agent into the user's world is exemplified by Thórisson's social agent Gandalf (Thórisson 1997, 2002), which modeled real-time, coordinated psycho-social dialogue skills including turn-taking, body posture, gaze, and speech.

An example of bringing the user into the agent's world is the research by Rickel et al. (2001). In their work, a virtually embodied character is situated in simulated training sessions for the army; the user is placed in a virtual world with the agent. The authors acknowledge that much improvement is still needed in areas of virtual human bodies, natural language understanding, emotions, and humanlike perceptions, yet a very small fraction of ongoing development in this area makes use of, or makes available, readily available components that can be plugged into other systems.

A third alternative exists for bringing together agent and user. The term *augmented reality* has been used to refer to systems that combine computer-generated virtual objects with the real surrounding scene in an interactive, real-time manner (Azuma 1997). Participants in an augmented reality environment typically wear either a semitransparent or a video see-through head-mounted display (HMD) that enables them to see localized virtual, graphical information rendered in context in the real environment. Placing autonomous characters into such augmented reality presents a compelling and powerful new human-computer interaction metaphor. Such characters can have many practical applications: They can enhance the realism of multiplayer games and virtual worlds, they can function as information retrieval assistants, and they could serve as virtual tour guides in real museums.

Mind Models

There has been substantial work in the area of mind construction for simulated humans (recent overviews can be found in McKevitt et al. 2002; and Granström, House, and Karlsson 2002). Given several decades of AI research, the first thing we'd like to avoid is reinventing the wheel. With its operators and problem spaces, Soar (Newell 1990) presents a highly unified set of principles and architectural constructs that address a number of psychological regularities observed in human cognition. The Soar architecture has been explored in various systems, including computer games (Laird and van Lent 2001, Laird 2002). Originally Soar focused on the concept of cognition in a narrow sense: perception and action were left out of the initial model. Yet to use Soar in simulated worlds requires that these gaps be bridged. Soar has since been extended to these areas, but its small set of cognitive principles don't make it very well suited as a foundation for studying cognitive model alternatives. Because of uniform granularity, Soar is also relatively complicated to use when building large systems. For that a methodology with better abstraction support is called for.

Behavior-based AI (compare with Bryson 2001, Brooks 1991) was a response to old-school knowledge representation and its use of explicit mental models of the world. The approach taken in behavior-based AI is to tightly couple the perception and action in integrated software structures that combine perceptual variables and processes with motor plans and their selection and execution. In a complex system, this can sometimes make software development and maintenance easier: implementing processes like limb-eye coordination can, for example, be done with simpler control structures than in most other approaches. However, these architectures tend to be limited to a single level of representation (compare with Maes 1990); they are not conducive to the acceptance of superstructures that are needed, for example, to do dialogue interpretation and generation at multiple levels of abstraction (although see Bryson 2000).

Puff the Magic LEGO Dragon (Bryson and Thórisson 2000) and the Ymir architecture be-

fore it (Thórisson 1996, 1999) are both examples of hybrid systems that use blackboards for modularizing information flow between processes based on type and scales of time. These systems contain both planners and schedulers, but their core is behavior-based, building upon a large number of modules that interact at different time scales. The system we describe in this article uses fewer modules, each module being a rather monolithic application, a choice made not for theoretical reasons but as a direct result of following our design principles for quickly building a relatively broad system. CDM can be applied in the design of systems incorporating any of the above approaches; moreover, to our knowledge, it is the best method for integrating principles from all of them.

Design Principles

Our approach is based on the concept of multiple interacting modules. Modularity is made explicit in our system by using one or more blackboards for message passing:

> To mediate communication between modules, use one or more blackboards with publish-subscribe functionality.

The blackboard provides a localized recording of all system events, system history, and state precisely at the level of abstraction represented in the messages. The benefits of message-based, publish-subscribe blackboard architectures are numerous. Among the most obvious is that their use allows the definition of modules whose input and output is fully defined by messages in the system, the messages thus embodying an explicit representation of the modules' contextual behavior. Building stateless modules wherever possible is therefore a direct goal in our approach, as it greatly simplifies construction. Given that the messages effectively implement the modules' application programming interfaces (APIs), they in turn directly mirror the abstraction level of the full system, making the blackboard architecture a powerful tool for incremental building and debugging of complex, interactive systems with multiple levels of abstraction. In many cases the messages we defined became key driving forces in the Mirage system, around which the modules were organized.

We also base our approach on the following meta-assumption:

> Only build functionality from scratch that is critical to the raison d'être of the system—use available software modules wherever possible.

This means that a critical part of the process is to define the project's goal, for example, whether the project is practically motivated (the primary goal being its functionality) or research-oriented (the primary goal is to answer one or more questions regarding mental process, human-computer interaction, or human-human interaction).

Steps

The specifics of CDM are encapsulated in the following outline. The indented text provides details by specific references to the example system that we built.

Step One. Define the Goals of the Project. Specify the primary motivation and goals behind the system.

> We wanted to show how to build an embodied, situated agent that significantly increased the value of an augmented reality by being able to interact naturally with human users. The primary focus was therefore on demonstrating the interaction between the user, the agent, and the environment and the integration of the technologies necessary to support such a system.

Step Two. Define the Scope of the System. Specify at a high level what the intelligent system is intended to do. Using narratives, write example scenarios that typify its behavior.

> We start with an initial write-up of a few high-level examples of user-agent interactions. From this, a more detailed specification of the abilities of the agent can be extracted, guiding the selection of which modules to include, such as gesture tracking, speech recognition, graphics, and so on. For example, in addition to including a module encapsulating knowledge about the objects in a room, we decided to include a news summarization system, Columbia Newsblaster (McKeown et al. 2002),[5] which enables the agent to tell us the world headlines when requested and read from the news articles.

Step Three. Modularize the System. The system is built using modules that communicate through a publish-subscribe mechanism or blackboards or a combination of both. Define the functional areas that the system must serve, and divide this into 5 to 10 distinct modules with roughly defined roles. The modules can then be recursively subdivided using the same approach (principle of divisible modularity). This step is best done in a group meeting where all developers can contribute to the effort.

First, *classify modules into one of three roles,* according to whether they serve perception, decision/planning, or action/animation. Each of these roles can be further split into more levels. For example, high-level perception versus low-level perception; short-, mid- and long-term planning; and high-level animation versus low-level animation.

Second, *find off-the-shelf software*. Locate software that is already implemented that could be used in the system. This step may require modification of the system's scope. (This step does not apply to components that are part of the project's raison d'être, as already explained.)

Third, *specify custom-written modules*. Having identified usable external modules, specify additional modules that need to be written and what their functionality should be.

Fourth, *follow the natural breaks in the information-processing path*. Let information flow via blackboards (1) when processes are of distinctly different types; (2) where information flow is relatively low frequency, (3) where information flow is event driven, or (4) where the method of information flow is pull rather than push, (that is, where the modules typically request their data, rather than having it "pushed" to them by other processes), provided that they are saving processing cycles by pulling (for example, because the frequency of data transfer via blackboards is lower that way). This step helps define the modules and their input and output.

Fifth, *define message types*. Specify how the various parts of the system should convey their state to the rest of the system. This step operationalizes the role of each module and defines the modules' interfaces. Define message content in such a way as to leave the modules as "raw processors" of data; in other words, make the blackboards (messages on the blackboards) contain all the data necessary for every module to produce its output. This pushes state information into the blackboards, greatly easing system construction. Where this is not possible, divide the data that needs to be external to the blackboards in a way that leaves a single module in charge of a clearly demarcated type of data so that the module serves as an interface or "wrapper" between the data and the blackboards. Should this not be possible, consider creating a separate module cluster with its own blackboard and recursively follow these guidelines.

Sixth, *avoid duplication of information*. If a module has been designated as being the only module in the system to need access to a particular type of data and then later another module is found to require access to the same data, several possibilities should be considered. Modules with highly different functional roles should typically not need access to the same set of data—if they do, it means they may have close functional dependencies; consider coupling them or putting them into the same executable with shared access to the same data. Perhaps more modules are found to need the data in question; consider creating a separate data store server with its own API that both modules can access, either via a blackboard or separate mechanism. Again, try to split the data such that only one module is in charge of one set of data.

Seventh, *determine the number of blackboards—which modules use which blackboards*. The most obvious reason for modules to share a blackboard is when they need to communicate directly, that is, when the output of one is the input of another. Consider using two or more blackboards if (1) there is a wide range of information types in the total set of messages in the system, for example, coexisting complex symbolic plan structures and simple Boolean switches; (2) there is a wide range of real-time requirements for the information flow, for example, high-frequency vision processing and low-frequency plan announcements; (3) the system needs to run on multiple computers to achieve acceptable performance. It is natural that modules with messages containing similar content share a blackboard. Some modules are a natural bridge between blackboards, as for example when a module producing decisions to act requires perceptual data to make those decisions. Blackboards serve both as engineering support and system optimization.

Step Four. Test the System against the Scenarios. Once a reasonably stable set of modules and messages has been reached, the scenarios outlined in step 2 are used to test-run the system on paper. Using these scenarios as a guide, every module should be made very simple at first, reading one type of message and posting another. A full end-to-end chain is specified for a single interaction case, enabling every element in the path to be tested on the whiteboard, along with the routing mechanisms, timing assumptions, and so on, very early in the design process. Such end-to-end testing should also be performed regularly during implementation. The viability of the modularization is verified, with regard to the following:

Expected communication (blackboard) through-

put. Network speed and computing power put natural constraints on the maximum throughput of the blackboard. For example, perceptual data, which may need to be updated at 10–15 Hz, may have to bypass central message paths. If the middleware/ blackboard is powerful enough to service a stream of data, it is preferable to route the data through it; a centralized data traffic center enables monitoring, debugging, prioritization, and so on. However, most of today's network solutions do not handle transmission of raw visual or auditory data at sufficiently high rates, necessitating workarounds. In practical terms, this means that one may need to combine what otherwise should naturally be two or more separate modules.

Efficient information flow. Static or semistatic information that is frequently needed in more than one place in the system may be a hint that the processes using that information should share an executable, thus containing the information in one place. Sometimes this is not a good idea, however, especially when the processes sharing the information are of very different natures, as in visual processing and motor control. In such cases it is better to set up a server (hardware or software) for the data that needs to be shared. Again, it is best that only one module be in contact with—or in charge of—data that cannot be represented in the content of messages for reasons of size, frequency of updating, or other reasons.

Convenience with regard to programming languages and executables. If two modules, for example speech synthesis and an animation system, are written in the same language, it may be convenient to put them together into one executable. This is especially sensible if the modules (1) use the same set of data, (2) are serially dependent on each other, (3) have similar update frequency requirements, (4) need to be tightly synchronized, or (5) are part of the same conceptual system (for example, low- and high-level visual perception). When conceptually distinct modules are combined into the same executable, care must be taken not to confuse their functionality in the implementation, which could prevent further expansion of either module in the future.

The ability to verify timing is especially important when building communicative characters since the system is bounded by real-time performance requirements: the agent needs to act in real time with regard to the user's natural behavior and expectations. If any part of the path, from input (such as a user finishing a question) to the agent responding (such as providing an answer), takes longer than the designers expect, or if some delay expected to be acceptable to the user turns out to be unacceptable, a redesign or significant modification may be required.

Three real-world examples serve to illustrate the implications of efficient information flow and convenience with regard to programming languages and executables. First, team members could not agree on how to design the perception part: should perception be part of the world, or should it be a separate module that was fed information continuously from the world module? Second, integration of perceptual information from speech and vision—how should it be implemented? A third question, which didn't come up until we had iterated the design a few times, was the placement of the speech synthesis module. Because this module was written in Java (we used FreeTTS[6]) and the world was written in Java 3-D,[7] we decided to put the speech module into the world (see figure 2) so that it could be better synchronized with the multimodal behavior of the agent's body (we had the animation scheduler control it because it gave us forced synchronization with almost no overhead). Even though the speech synthesizer contains knowledge, prosody generation, which rightfully belongs to the "mind" rather than the body, we decided in favor of performance over theoretical correctness.

Step Five. Iterate. Go through steps 2–4 as often as needed.

Step Six. Assign the Modules to Team Members. The natural way to assign responsibilities is along the lines of the modules, following people's strengths and primary interest areas. Every module or module cluster gets one lead team member who is responsible for that module's or module cluster's operation, for defining the messages the module or module cluster receives and posts. A good way to assign tasks, especially if the number of team members is small, is to borrow from extreme programming[8] and assign them to pairs: one lead, chosen for his or her primary strength, and one support, chosen by his or her interest or secondary strength. This step depends ultimately on the goals of the project and the experience and goals of the project members.

We had eight modules and five team members; members served as lead on two

modules and support on one, or lead on one and support on two.

Step Seven. Test All the Modules in Cooperation. It is critical to test the modules at run time early on in the implementation phase to iron out inconsistencies and deficiencies in the message protocols and content. Selecting the right middleware for message routing and blackboard functionality is key, as the final system depends on this being both reliable and appropriate for the project. Open adapters have to exist for the middleware to allow integration of new and existing modules. The testing is then done with the chosen middleware and stub versions of the modules before their functions have been implemented. To begin with, the stubs can simply print out the messages they take as input. They can also be set to publish canned messages via keyboard or graphical menu input. Using these two features, the stub modules can be used to do simple integration testing of the entire system.

Integration testing often takes longer than expected. It is also one of the most overlooked steps, yet it cannot be avoided. Doing so only forces integration testing to happen later in the process. Later testing, in turn, delays an accurate estimate of the project's full scope and increases the probability of missed deadlines and cancellation of planned functionality.

Step Eight. Build the Modules to Their Full Specification. This step is naturally done in frequent alternation with the prior step; as functionality is added, regression errors tend to creep in. However, designing modules with clearly defined set of inputs and outputs dramatically decreases the amount of potential communication errors and makes the system's behavior more predictable. In addition to building the full modules, one must build into each module extensive fine-tuning capabilities, such as the timing of message posting and tracking of state and history, to achieve the desired communication between modules later on.

Step Nine. Tune the System with All Modules Cooperating. It is important to test the system for a while with a group of people using it and to tune it based on the results. Especially for real-time interactive systems, variations in people's interaction style often unearth faults in the implementation logic, especially simplifications that the team may have made in the interest of time, and since the system is based on multiple modules interacting and cooperating, their real performance is not understood until they have been subjected to a number of different user input scenarios. This step can be arbitrarily long, depending on the complexity of the interaction between modules and the complexity of the message content being transmitted between them.

We did very little tuning of our system, but as a result, we did not implement all the functionality we had originally planned.

Execution in a Classroom Setting

In a classroom setting we divide the above methodological principles into four phases. The first phase starts off with a single-paragraph proposal from each student.. The proposals, which are reviewed and approved by the instructor, should outline the topic, motivation, and components of the system to be built. When a project has been chosen (sometimes by merging two or more ideas into one system), the second phase focuses on creating an integration plan, the expected results, and a final (functional) form of the implemented system. It is important to write usage scenarios exemplifying the major functional aspects of the desired system.

During the third phase each team member writes up a description and breakdown (as long and detailed as necessary) of his or her individual contribution to the project, identifying potential difficulties and challenges and proposing solutions to deal with them, as well as identifying which team members he or she needs to collaborate with. The description details the modules and the functions that are to be coded, along with the input and output messages that the modules need and which modules produce or consume them. The more detail about the messages' content and syntax in these write-ups, the better for the whole team, because the write-ups help concretize how the whole system works. At the end of this phase, the team should have a project plan with measurable milestones, based on the outline from phase two.

The fourth phase is the implementation. The lead team members for each part of the system, in collaboration with the instructor, help revise the project, reassign tasks, and so on, as necessary as the project progresses. The leads are ultimately responsible for making sure that all modules in the project can handle minimal interaction, as defined.

Mirage System

We now give an overview of the system used for testing the methodology to provide an example system built with the methodology and better equip readers to evaluate its applicability in their own context.

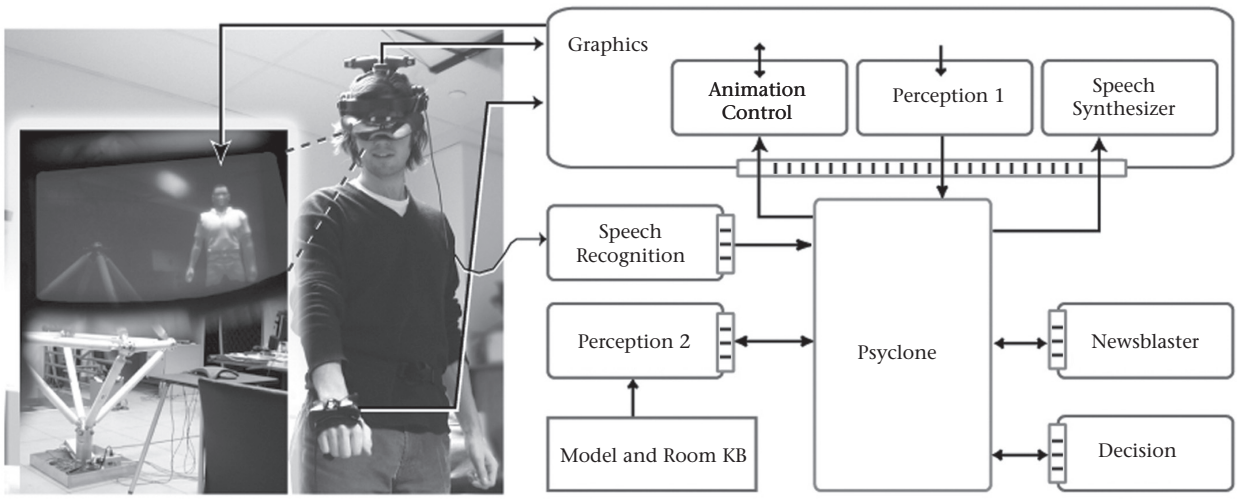


Figure 2. Mirage System Architecture, Showing Sensor and Display Connections to the Physical Devices.
Arrows to and from Psyclone indicate message and control flow. Small striped rectangles on the edges of modules signify Psyclone adapters.

To implement publish-subscribe functionality between modules we used CMLabs' Psyclone,[9] a platform that provides easy hookup via Java and C++ adapters. Psyclone shares some features with earlier efforts in this area, for example Cohen et al.'s (1994) Open Agent Architecture and Fink and colleagues' (1996) DACS. In our case the blackboard's state and history served primarily as support for iterative development and debugging. The architecture of the Mirage system is shown in figure 2; a list of the modules is given in table 1. We found a single blackboard sufficient for the level of complexity of our system but would probably advocate a multiblackboard approach with further expansion. The system and its modules are discussed briefly in the following sections.

Augmented Reality Environment

To bridge the gap between the real and the virtual and bring our agent to "life," we chose an augmented reality approach. A head-mounted stereoscopic display (HMD) with selective transparency allows us to render the agent as a 3-D, stereoscopic overlay graphic that the user can perceive to be situated in the real environment. The glasses are fully transparent except where the agent appears. Special tracking technology makes the agent appear just like any other object with a fixed location in the room.

The augmented reality environment consists of both perception and action/animation scheduling modules, as well as a complete graphics system that renders our embodied agent (figure 1). The user wears the optical see-through glasses (Sony LDI-D100B) and sees the real world overlaid with a stereoscopic image of the agent generated in real time. Using position and orientation tracking of the user's head and hand,[10] the system can make the virtual agent appear as an active part of the real environment. Mirage can remain stationary, or move around in the world, independently of the user, giving the impression that he occupies an actual physical location in space.[11]

Our system has a detailed 3-D model of the entire room in which Mirage is placed, including sofa, tables, chairs, desks, computers, and so on, giving him a certain amount of knowledge about his surroundings. Mirage can thus "perceive" the real objects in the room, walk around the (real-world) furniture and objects, and orient himself relative to these. All graphics are implemented using the Java 3-D API. While the 3-D model of Mirage and the animation schedulers for him were designed and implemented as part of this project, the tracking and graphical model of the environment had already been built and had to be adapted to our system.

Interaction Capabilities

We see future versions of Mirage as guiding tours in museums or tutoring employees about the layout of production plants, giving him potential as part of future commercial applica-

tions. To make the agent interactive and conversational, we implemented a multimodal input system that uses speech recognition and motion tracking of the user's right hand. The agent itself is capable of multimodal communication with the user via a speech synthesizer, body language, and manual gesture.

For speech recognition we used the Sphinx speech recognizer from Carnegie Mellon University.[12] Mirage has access to a knowledge base for all the objects where type, use, names, and other attributes of individual objects are stored, allowing him to talk about them. The user can point at any object in the room and ask Mirage what that object is. When asked about the news today, Mirage recites news summaries generated dynamically by the Newsblaster system from online news feeds (McKeown et al. 2002). Mirage is also aware of proximity, and it knows its own name; when either his name is spoken or the user comes within 1 meter, Mirage will turn to and greet the user.

Perception, Decision, and Action

To allow for the human interaction scenarios mentioned above, our system used two custom-written perception modules in combination with a custom decision module. The role of the perception modules is to create messages about changes in the environment, including recognition of the user's speech, proximity of the user to the Mirage agent, and so on. Those messages are then used by the Decision module to create sensible responses to the user's input that are subsequently executed by the animation scheduler. Examples of messages between modules are given in table 2.

The Perception-1 module is built into the augmented reality environment. It keeps track of the agent's and the user's positions and orientations, noting changes of states such as "is the agent visible to the user," "is the agent looking at the user," and so on. Perception-1 uses technology based on previous work by Olwal, Benko, and Feiner (2003) in which *SenseShapes* (volumetric regions of interest) are used to determine the user's selection and pointing. This relatively simple selection mechanism is triggered on any speech command, returning a list of objects being pointed to at the time of speech. The output of Perception-1 is further processed by the Perception-2 module, where objects pointed at are combined with speech based on event time stamps, to create a holistic perception of the user's action. Implied meaning of perceptions is not represented explicitly but emerges from their combination with decision mechanisms, thus leaving out the use of an explicit "cognition" module. In a more sophisticated humanoid, we envision several separate cognitive modules for tasks such as reasoning, dialog management, and so on all interacting via message passing.

The primary reason for separating perception into two modules was to be able to use the built-in geometric computation methods of the graphics subsystem and its already-available ability to detect selection of objects, pointing, and intersection with other objects. Doing those checks outside of the graphics environment would have been inefficient and harder to implement, albeit theoretically cleaner. This meant that perceptual responsibilities were eventually split between two modules (not counting speech). However, once those basic messages regarding geometric operations were created, integrating those with the speech messages coming separately from the speech recognizer module was a relatively easy task, handled through the separate integration module, Perception-2.

The Decision module is responsible for integrating and interpreting all messages from the perception modules and forming a response action for the agent. That selected action is then sent to the Action Scheduling module, which coordinates the avatar animation and speech synthesis. When the Decision module receives contradicting inputs or is unable to settle on a single top-level decision, it can request that the avatar ask questions for clarification.

Discussion

Two of the third-party components we used turned out to function inadequately in our final system. Sphinx II was too difficult to set up and tune properly in the time we had planned. We could have gotten better accuracy with minimal tuning using an off-the-shelf speech-recognition product such as ViaVoice or Dragon Naturally Speaking. We also found that the speech produced by FreeTTS is very hard to understand in noisy environments—people who tried our demo had a very hard time understanding much of what Mirage said, save for the very shortest, most obvious utterances. While it is great to have an open source synthesizer in Java, it would be even better if it generated more comprehensible output.

One potential problem of using existing modules is that sometimes they lack control from the outside. Although such limitations turned out not to be a problem with Mirage (we had a clear idea up front of how to engineer our way around them), the issue is a generic one that will not go away. As a future remedy to this problem we are working on a set of principles to help guide module builders

when selecting, documenting, and exposing the modules' "knobs and dials" so that modules are better suited for use with each other in multimodule, multiauthor systems.

Psyclone was a good match to CDM and made it easy to distribute processes between machines, something that enabled us to achieve acceptable run-time performance quickly. This approach of distributing processes is likely to become increasingly attractive over the coming years as computer prices keep dropping, making "garage-AI" feasible and bringing new developers to the field with clusters of cheap computers from yesteryear, networks of which can supply the processing power even for serious AI work. The computer industry has no motivation to promote the reuse of old computing equipment; software such as Psyclone does, and Mirage shows this to be a promising option.

Unlike closed virtual-world systems, ours is an open loop, with unpredictable human behavior streaming in real time. Since the full system lives on separate computers that cannot be run in lock-step, unpredictable delays on the network and the computers can differentially affect the performance of the various modules. This calls for an architecture in which modules have tolerance for the error caused by such delays. Although we did not take the implementation very far in this direction, our general approach reflects these constraints directly. For example, universal time stamps were used extensively throughout to coordinate events and messages between modules.

Module	Posts	Subscribing Modules
Speech Recognition	Input.Perc.UniM.Hear:Off Input.Perc.UniM.Hear:On Input.Perc.UniM.Voice.Speak	Perception-1 Perception-2
Perception-1	knowledge.user.agent-proximity knowledge.user.looking-at-agent knowledge.agent.looking-at-user knowledge.user.looking-at-model knowledge.model.visible knowledge.user.pointing-at-model-objects knowledge.user.pointing-at-room-objects	Perception-2
Perception-2	knowledge.dialog-topic knowledge.dialog-topic.shifting knowledge.user-goal	Decision

Table 1. A List of Selected Modules and Their Posted and Subscribed-to Message Types.

Message Type	Input.Perc.UniM.Hear:On
Sending Module	Speech
Subscribing Modules	Perception-1, Perception-2
Description	Speech started
Arguments	None
Message Type	Input.Perc.UniM.Hear:Off
Sending Module	Speech
Subscribing Modules	Perception-1, Perception-2
Description	Speech stopped
Arguments	None

Table 2. Example Specification of Message Format for Modules.

Conclusion

Using CDM we designed and implemented a conversational character embodied in an augmented reality system. We developed and incorporated a variety of technologies to serve our goal of a natural and simple human-humanoid interaction that worked in real time. The design and implementation of the full system took place within a period of nine weeks, with five students dividing the work evenly, taking an estimated total of two mind-months ("man"-months) to complete. The results demonstrate that the methodology and its modularization principles work very well for the construction of such systems.

The student team was composed of one undergraduate and four second-year master's students. Before starting the Mirage project, only the instructor had experience with the methodology, and none of the students had built an interactive system of this scale before. One student had more than what could be considered a minimal experience in building interactive systems, while two had prior experience with speech-recognition systems. None of them were AI majors—one had taken no introductory AI courses; the others had taken one. Students felt that the approach taken gave them an increased overview of the system's functioning—something they would not have experienced otherwise. We feel confident to conclude that the methodology works well for building large, modular systems, even for teams on which most of the members are novices in AI. This makes CDM an excellent choice for classroom use.

Because CDM was developed for integrating large numbers of modules and system hierarchies, it is not sensible to apply it to the reinvention of well-tested algorithms already developed over the past decades, such as, for example, natural language parsing. However, the approach's utility is clear for incorporating those available algorithms into larger systems, where their operation can be observed in context and where they have some hope of achieving utility. Mirage provides examples of how this can be done.

Explicit representation of module interaction through messages, which can be inspected both on paper and at run time, increases the system's transparency and increased all team members' understanding of how the various parts of the system work. In spite of the typical optimism of (young) programmers, the students were surprised that all modules, except for the Speech Recognition module, were working as planned at the end of the semester (the project did not start until the last third of the semester). We feel the total estimated development time of two mind-months strongly supports the hypothesis that CDM speeds up system creation. A parallel can be found in the work by Maxwell and colleagues (2001), who explored the use of a message-passing architecture to facilitate modular integration of several key technologies needed for constructing interactive robots. Their robots have sophisticated vision, planning and navigation routines, speech recognition and synthesis, and facial expression. They achieved impressive results with limited resources: ten undergraduates working on the project for eight weeks, constructing three robots with this architecture, two mobile and one stationary.

One question that might be raised is whether the system thus built is perhaps too simple, somehow hiding a hypothetical sharp rise in difficulty when applied to the design of more realistic models of mind, with one or two orders of magnitude greater complexity (or requiring functionalities not addressed in Mirage). Although we cannot answer this conclusively, we feel that evidence from earlier variants of the methodology counter this claim, it having been used with good results in, for example, the design of a large virtual-reality environment (Bryson and Thórisson 2000) and a natural language system deployed for CNET and Buy.com (Lucente 2000) that supported large numbers of Internet users. The methodology is very general; there is no doubt that systems of significantly greater complexity than Mirage can be built using the approach, reaping the same—quite possibly even greater—benefits.

In summary, we have presented (1) a novel, relatively broad, real-time system developed by a small group of relatively inexperienced students working within a very short period of time; (2) team members' significantly heightened awareness of the functional breadth needed to construct interactive intelligent systems; (3) integration of prior work into a new system, and (4) a novel demo with commercial potential, as well as commercial examples of earlier uses of the methodology, showing that it can help in the development of real-world AI systems. While we don't consider the Mirage project to provide a scientific test of the hypothesis that models of mind can be developed in a modular fashion, nothing in our explorations indicates some sort of limit to the kinds of mental structures that CDM could support. The approach applies as equally to building "believable" interaction as to the goal of accurately modeling psychological and psycho-biological processes. Indeed, since it allows modularization to happen at any size, and at one, two, or many, levels of detail, there is hardly a system to which the methodology cannot be applied. That does not mean to say that it is equally applicable to modeling all kinds of systems—indeed, we believe that it is most applicable for systems with ill-defined boundaries between subsystems and where the number of variables to be considered is relatively large. In the current state of science, primary examples include ecosystems, biological systems, social systems, and intelligence.

We are currently in the process of setting up a forum, in collaboration with a larger team, to further develop the constructionist approach to AI and build a repository of "plug-and-play" AI modules and systems. We invite anyone with interest to join in this effort; the URL is mindmakers.org.

Acknowledgments

We thank Steven Feiner for access to his laboratory, technology, and students; Alex Olwal for creating the avatar; the Columbia Newsblaster team; our reviewers for excellent suggestions; and CMLabs for Psyclone, a free academic version of which can be downloaded from www.mindmakers.org/projects/Psyclone. Psyclone is a trademark of Communicative Machines Laboratories. LEGO is a trademark of The LEGO Group, A/S.

Notes

1. While Newell's architecture, Soar, is based on a small set of general principles that are intended to explain a wide range of cognitive phenomena, Newell makes it very clear in his book (Newell 1990) that he does not consider Soar to be *the* unified theory of cognition. We read his call for unification not to mean in the narrow sense the particular premises he chose for Soar but rather to refer in the broader sense to the general breadth of cognitive models.

2. www.mindmakers.org/air/airPage.jsp

3. Personal communication.

4. If a screen-based agent is aware of real-world objects, as well as virtual ones, it might be classified as an augmented reality system. Traditionally, though, such classification would also imply that the user's reality is augmented in some obvious way by the agent/system.

5. www.cs.columbia.edu/nlp/newsblaster

6. freetts.sourceforge.net/docs/index.php

7. java.sun.com/products/java-media/3D/

8. www.extremeprogramming.org/

9. See The Psyclone Manual (www.cmlabs.com/psyclone). CMLabs Psyclone executables can be downloaded for Linux, Windows, and Macintosh OSX from www. mindmakers.org/projects/psyclone.

10. InterSense IS900. http://www.isense.com/

11. The tracking keeps the agent still most of the time; fast movements of the head will cause the agent's image to jiggle.

12. www.speech.cs.cmu.edu/sphinx/

References

Azuma, R. T. A Survey of Augmented Reality. 1997. *Presence: Teleoperators and Virtual Environments*. 6(4): 355–385.

Badler, N.; Palmer, M. S.; and Bindiganavale, R. 1999. Animation Control for Real-Time Virtual Humans. *Communications of the ACM* 42(8): 65–73.

Bakker, B.; and den Dulk, P. (1999). Causal Relationships and Relationships between Levels: The Modes of Description Perspective. In *Proceedings of the Twenty-First Annual Conference of the Cognitive Science Society,* 43–48, ed. M. Hahn and S. C. Stoness. Mahwah, N.J.: Lawrence Erlbaum Associates.

Bischoff, R. 2000. Towards the Development of "Plug-and-Play" Personal Robots. Paper presented at the First IEEE-Robotics and Automation Society International Conference on Humanoid Robots, Cambridge, Mass., September 7–8.

Bischoff, R.; and Graefe, V. 1999. Integrating Vision, Touch and Natural Language in the Control of a Situation-Oriented Behavior-Based Humanoid Robot. In *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics,* 999–1004. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Brooks, R. A. 1991. Intelligence without Reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence,* 569–595. San Francisco: Morgan Kaufmann Publishers.

Brooks, R. A.; Ferrell, C.; Irie, R.; Kemp, C. C.; Marjanovic, M.; Scassellati, B.; and Williamson, M. 1998. Alternative Essences of Intelligence. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and the Tenth Conference on Innovative Applications of Artificial Intelligence,* 961–968. Menlo Park, Calif.: AAAI Press.

Bryson, J. 2001. Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. Ph.D. diss., Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Mass..

Bryson, J. 2000. Making Modularity Work: Combining Memory Systems and Intelligent Processes in a Dialog Agent. Paper presented at the Society for the Study of Artificial Intelligence and the Simulation of Behaviour (AISB) 2000 Symposium on How to Design a Functioning Mind, Birmingham, England, April 17–20.

Bryson, J.; and Thórisson, K. R. 2000. Bats, Dragons and Evil Knights: A Three-Layer Design Approach to Character-Based Creative Play. *Virtual Reality* (Special Issue on Intelligent Virtual Agents) 5(1): 57–71. Heidelberg: Springer-Verlag.

Cohen, Philip R.; Cheyer, Adam J.; Wang, Michelle; and Baeg, Soon Cheol. 1994. An Open Agent Architecture. In *Readings in Agents,* 197–204, ed. M. N. Huhns and M. P. and Singh. San Francisco: Morgan Kaufmann Publishers.

Fink, G. A.; Jungclaus, N.; Kummer, F.; Ritter, H.; and Sagerer, G. 1996. A Distributed System for Integrated Speech and Image Understanding. In Proceedings of the Ninth International Symposium on Artificial Intelligence. Monterrey, Mexico: Instituto Tecnologico y de Estudios Superiores de Monterrey.

Fink, G. A.; Jungclaus, N.; Ritter, H.; Saegerer, G. 1995. A Communication Framework for Heterogeneous Distributed Pattern Analysis. In Proceedings of the First IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP–95), 881–890 Piscataway, N.J.: : Institute of Electrical and Electronics Engineers, Inc.

Granström, B.; House, D.; and Karlsson, I., eds. 2002. *Multimodality in Language and Speech Systems*. Dodrecht, The Netherlands: Kluwer Academic Publishers.

Iba, S.; Paredis, C. J. J.; and Khosla, P. K. 2002. Interactive Multi-Modal Robot Programming. In Proceedings of the 2002 IEEE International Conference on Robotics and Automation. Piscataway, N.J.: Institute of Electrical and Electronics Engineers, Inc.

Laird, J. 2002. Research in Human-Level AI Using Computer Games. *Communications of the ACM* 45(1): 32–35.

Laird, J. E.; and van Lent, M. 2001. Human-Level AI's Killer Application: Interactive Computer Games. *AI Magazine* 22(2): 15–25.

Lucente, M. 2000. Conversational Interfaces for E-Commerce Applications. *Communications of the ACM* 43(9): 59–61.

Maes, P. 1990. Situated Agents Can Have Goals. In *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back,* 49–70, ed. P. Maes. Cambridge, Mass.: The MIT Press.

Martinho, C.; Paiva, A.; and Gomes, M. R. 2000. Emotions for a Motion: Rapid Development of Believable Pathematic Agents in Intelligent Virtual Environments. *Applied Artificial Intelligence* 14(1): 33–68.

Maxwell, B. A.; Meeden, L. A.; Addo, N. S.; Dickson, P.; Fairfield, N.; Johnson, N.; Jones, E. G.; Kim, S.; Malla, P.; Murphy, M.; Rutter, B.; and Silk, E. 2001. REAPER: A Reflexive Architecture for Perceptive Agents. *AI Magazine* 22(1): 53–66.

McCarthy, J.; Minsky, M.; Sloman, A.; Gong, L.; Lau, T.; Morgensten, L.; Mueller, E. T.; Riecken, D.; Singh, M.; and Singh, P. 2002. An Architecture of Diversity for Common Sense Reasoning. *IBM Systems Journal* 41(3): 524.

McKeown, K. R.; Barzilay, R.; Evans, D.; Hatzivassiloglou, V.; Klavans, J. L.; Nenkova, A.; Sable, C.; Schiffman, B.; and Sigelman, S. 2002. Tracking and

Summarizing News on a Daily Basis with Columbia's Newsblaster. Paper presented at the Conference on Human Language Technology (HLT '02), San Diego, Calif., Mar. 24–27.

McKevitt, P.; Ó Nulláin, S.; and Mulvihill, C., eds. 2002. *Language, Vision and Music*. Amsterdam: John Benjamins.

Minsky, M. 1986. *The Society of Mind*. New York: Simon and Schuster.

Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, Mass.: Harvard University Press.

Nicolesco, M. N.; and Mataric, M. J. 2002. A Hierarchical Architecture for Behavior-Based Robots. In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems. New York: Association for Computing Machinery.

Olwal, A.; Benko, H.; and Feiner, S. 2003. SenseShapes: Using Statistical Geometry for Object Selection in a Multimodal Augmented Reality System. In Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality. Los Alamitos, Calif.: IEEE Computer Society.

Rickel, J.; Gratch, J.; Hill, R.; Marsella, S.; and Swartout, W. 2001. Steve Goes to Bosnia: Towards a New Generation of Virtual Humans for Interactive Experiences, In *Artificial Intelligence and Interactive Entertainment: Papers from the 2000 AAAI Spring Symposium,* ed. John Laird and Michael van Lent. Technical Report SS-01-02. Menlo Park, Calif.: American Association for Artificial Intelligence.

Simmons, R.; Goldberg, D.; Goode, A.; Montemerlo, M.; Roy, N.; Sellner, B.; Urmson, C.; Schultz, A.; Abramson, M.; Adams, W.; Atrash, A.; Bugajska, M.; Coblenz, M.; MacMahon, M.; Perzanowski, D.; Horswill, I.; Zubek, R.; Kortenkamp, D.; Wolfe, B.; Milam, T.; and Maxwell, B. 2003. GRACE: An Autonomous Robot for the AAAI Robot Challenge. *AI Magazine* 24(2): 51–72.

Terzopoulos, D. 1999. Artificial Life for Computer Graphics. *Communications of the ACM* 42(8): 33–42.

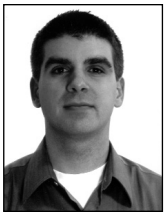
Thórisson, K. R. 2002. Natural Turn-Taking Needs No Manual: Computational Theory and Model, from Perception to Action. In *Multimodality in Language and Speech Systems,* 173–207, ed. B. Granström, D. House, and I. Karlsson. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Thórisson, K. R. 1999. A Mind Model for Multimodal Communicative Creatures and Humanoids. *International Journal of Applied Artificial Intelligence* 13(4–5): 449–486.

Thórisson, K. R. 1997. Gandalf: An Embodied Humanoid Capable of Real-Time Multimodal Dialogue with People. In Proceedings of the First ACM International Conference on Autonomous Agents. New York: Association for Computing Machinery..

Thórisson, K. R. 1996. Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills. PhD diss., Massachusetts Institute of Technology Media Laboratory, Cambridge, Mass.

Kris Thórisson has been developing AI systems and related technologies for 15 years. As a codirector of the Wizard Group at LEGO Digital he and his group developed a large virtual world inhabited by interactive creatures. At the Massachusetts Institute of Technology, he created Gandalf, a graphical guide to the solor system featuring multimodal perception and action, real-time turn-taking, and task knowledge. As vice president of engineering at Soliloquy Inc. he directed the development of on-line natural language bots that assisted customers of CNET and Buy.com. His recent startup, Radar Networks Inc., develops semantic technologies for next-generation Web services. Thórisson has consulted on business and technology for numerous companies including British Telecom, Interactive Institute, Kaupthing Investment Bank, and NASA. He is currently chief technology officer of Radar Networks Inc., a company he cofounded. He has degrees in psychology and human factors, and holds a Ph.D. in media arts and sciences from the Massachusetts Institute of Technology Media Laboratory. Thórisson recently joined the faculty at faculty at Reykjavik University's Department of Computer Science, where he develops communicative humanoids. He can be reached at kris@@ru.is



Hrvoje Benko is currently a Ph.D. student working with Steven Feiner at the Department of Computer Science at Columbia University, New York. Among his research interests are hybrid user interfaces, augmented reality, and wearable mobile devices. He is a member of the Institute of Electrical and Electronics Engineers. His e-mail address is benko@cs.columbia.edu.

Andrew Arnold is a software engineer at Google in New York City. He received a B.A. in computer science from Columbia University and began his Ph.D. in machine learning at Carnegie Mellon University in 2004. His primary research interests are unsupervised machine learning and anomaly detection. He can be reached at aoa5@columbia.edu.

Sameer R. Maskey is a Ph.D. student at Columbia University. His primary area of research is speech summarization. He has been working with Julia Hirschberg in exploring structural, linguistic, and acoustic features to summarize broadcast news. He is also interested in machine learning techniques for mining speech data. He can be reached at srm2005@columbia.edu.

Denis Abramov holds an M.Sci. degree in computer science from Columbia University and obtained his B.A. from Brandeis University. He has done speech-recognition research at Dragon Systems as well as consulting for startup companies. For the past few years he has been developing applications for the financial industry. His email is daa82@columbia.edu.

Aruchunan Vaseekaran is vice president for RandD at m-Rateit.com, a startup firm that he cofounded. This firm develops intelligent expert-based solutions for personal health management. He holds a B.Sc. in electrical engineering and an M.Sc. in computer science from Columbia University. He is a member of Tau Beta Phi, the National Engineering Honor Society. Vaseekaran can be reached at vasee@ ontash.net.