To appear in: A.I. Magazine, fall, 2004.

Constructionist Design Methodology for Interactive Intelligences

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

Abstract

We present a methodology for designing and implementing interactive intelligences. The *Constructionist Methodology* – so called because it advocates modular building blocks and incorporation of prior work – addresses factors that we see as key to future advances in A.I., including interdisciplinary collaboration support, coordination of teams and large-scale systems integration.

We test the methodology by building an interactive multi-functional 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 function, as well as more abstract topics such as current news, through situated multimodal dialog.

The results demonstrate the Constructionist Methodology's strength in simplifying the modeling of complex, multi-functional systems requiring architectural experimentation and exploration of unclear sub-system boundaries, undefined variables, and tangled data flow and control hierarchies.

Introduction

The creation of embodied humanoids and broad A.I. systems requires integration of a large number of functionalities 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, communicating 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 paper we present the design methodology and describe the system that we built to test it.

Newell [1992] urged for the search of unified theories of cognition[1], and recent work in A.I. has increasingly focused on integration of multiple systems (cf. [Simmons et al. 2003, McCarthy et al. 2002, Bischoff et al. 1999]). Unified theories necessarily

[1] While Newell's architecture Soar is based on a small set of general principles, intended to explain a wide range of cognitive phenomena, Newell makes it very clear in his book [Newell 1992] that he does not consider Soar to be *the* unified theory of cognition. We read his call for unification not in the narrow sense to mean the particular premises he chose for Soar, but rather in the more broad sense to refer to the general breadth of cognitive models.

mean integration of many functionalities, but our prior experience in building systems that integrate multiple features from artificial intelligence and computer graphics [Bryson & 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 A.I. person, and vice versa.

Several factors motivate our work. First, a much-needed move towards building on prior work in A.I., to promote incremental accumulation of knowledge in creating intelligent systems, is long overdue. The relatively small group who is working on broad models of mind, bridging across disciplines, needs better ways to share results and work together, and to work with others outside their field. To this end our principles foster re-usable software components, through a common middleware specification,[2] and mechanisms for defining interfaces between components. Second, by focusing on the re-use 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, by bridging across multiple functionalities in a single, unified system, researchers' familiarity and breadth of experience with the various models of thought to date – as well as new ones – increases. 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 A.I.* – puts modularity at its center: Functionalities of the system are broken into individual software modules, which are typically larger than software classes (i.e. 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 information content 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 functionalities.

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 its 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

[2] http://www.MINDMAKERS.ORG

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.

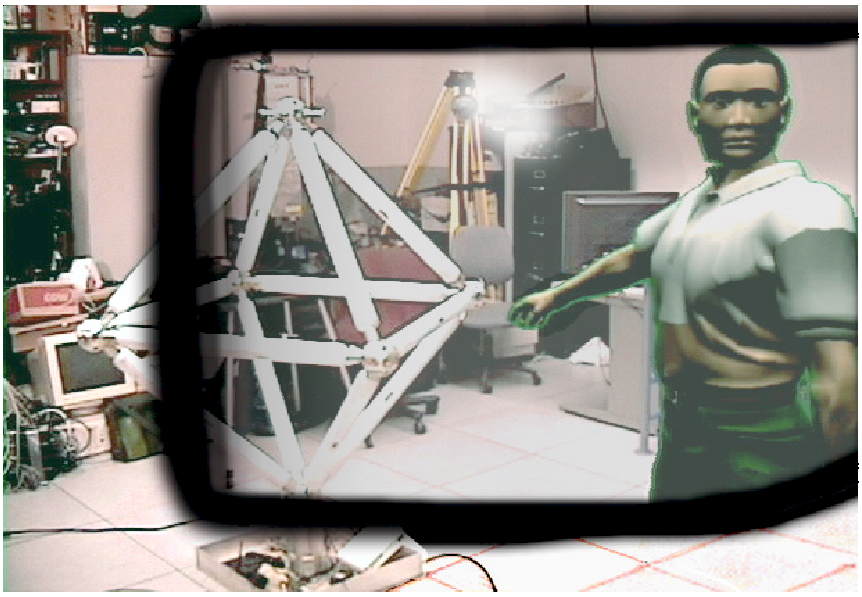

Figure 1: Our embodied agent Mirage is situated in the lab. Here we see how he appears to the user through the head-mounted glasses. (Image has been enhanced for clarity.)

There is essentially nothing in the Constructionist approach to A.I. that lends it more naturally to behavior-based A.I. [c.f. Brooks 1991] or "classical" A.I. – its principles sit beside both. In fact, since Constructionist A.I. 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 them navigate the real-world environment. The character, called Mirage, should appear as a transparent, ghost-like stereoscopic 3-D graphic superimposed on the user's real world view (Figure 1). This

[3] Personal communication, 1994.

system served as a test-bed for our methodology; it is presented in sufficient detail here to demonstrate the application of the methodology and to show its modular philosophy, which it mirrors closely.

Related Work

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 et al. [2000] describe an architecture designed to facilitate modular, rapid development of theatrical agents in virtual worlds. Like many other systems [cf. Granström et al. 2002, Laird 2002, McKevitt et al. 2002, Badler et al. 1999, Terzopolous 1999], their agents live in an enclosed virtual world where the update cycle of the world can be artificially controlled. Martinho et al.'s architecture involves splitting the world, along with the agents, into three separate modules, each which can be developed independently of the others. As in our approach, this enables parallel execution of implementation, reducing 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 et al.'s impressive HERMES robot [2000]. Simmons et al's robot Grace [2003], which involved the work of 5 institutions and over 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 & Thórisson 2000], and Behavior-Oriented Design [Bryson 2002]. Bryson [2002] details the elements of BOD, using principles from behavior-based A.I., 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 pre-existing modules. The Ymir architecture [Thórisson 1999] presents relatively general modularization principles, using module types such as perceptors, deciders and actions with subtypes, in a way that clearly distinguishes their role 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 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 virtual world simulations.

Due to their physical presence in the real world, robots are inherently embodied. There has been a recent surge in simulating human-like movement and decision-

[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.

making process in robotics, and even more recent have been attempts to create a goal-oriented human-like interaction between the robot and the user. An example is the work of Iba et al. [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 1996, 2002], which modeled real-time, coordinated psycho-social dialog skills including turn-taking, body posture, gaze and speech.

An example of the latter approach 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 human-like 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 which combine computer-generated virtual objects with the real surrounding scene, in an interactive, real-time manner [Azuma 1997]. Participants in an augmented reality typically wear either semi-transparent or video see-through head mounted display (HMD) that enables them to see localized and contextualized 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 multi-player 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 et al. 2002]). Given several decades of A.I. research, the first thing we'd like to avoid is re-inventing the wheel. With its operators and problem spaces, Soar [Newell 1992] 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 & 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 these gaps to 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 and explicit support for multiple levels or "modes of description" [Bakker & Dulk 1999] is called for.

Behavior-based A.I. (cf. [Bryson 2001, Brooks 1991]) was a response to the old-school knowledge representation, and its use of explicit mental models of the world. The approach taken in behavior-based A.I. is to tightly couple the perception and action in integrated software structures which 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 (c.f. [Maes 1990]); they are not very conducive to accept superstructures, which 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 & Thórisson 2000] and the Ymir architecture before it [Thórisson 1999, 1996] 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, interacting at different time scales. The system we describe in this paper uses fewer modules, each module being a rather monolithic application, a choice not made for theoretical reasons but a direct result of following our design principles for quickly building a relatively broad system. The Constructionist Methodology 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 ones is that it 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 state-less modules wherever possible is therefore a direct goal in our approach, as it greatly simplifies construction. Given that the messages effectively implement the modules' APIs, they in turn mirror directly the abstraction level of the full system, making the blackboard architecture a powerful tool for incremental building and de-bugging 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'etre 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(s), for example, is it practically motivated (primary goal being its functionality), or is it research-oriented (primary goal is to answer one or more questions regarding mental

process, human-computer interaction, or human-human interaction).

Steps

The specifics of the Constructionist Methodology are encapsulated in the following outline. Italicized sections provide details by specific references to the example system that we built.

1. **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.

2. **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 interaction. 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, etc. 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.

3. **Modularization.** The system is built using modules that communicate via a pub-sub mechcanism and/or blackboard(s). Define the functional areas that the system must serve, and divide this into 5-10 distinct modules with roughly defined roles. The modules can then be recursively sub-divided 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.

   a. **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.
   b. **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'etre, as explained above.)
   c. **Specify custom-written modules.** Having identified usable external modules, specify additional modules that need to be written, and what their functionality should be.

[5] http://www.cs.columbia.edu/nlp/newsblaster

d. **Follow the natural breaks in the information processing path**. Let information flow via blackboards when (i) processes are of distinctly different *types*; (ii) where information flow is relatively low-frequency, (iii) where information flow is event driven, and/or (iv) pull rather than push, (i.e. 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 (e.g. because the frequency of data transfer via blackboards is lower that way). This step helps define the modules and their input and output.
e. **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 their 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 data necessary for every modules to produce their 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, serving as an interface or "wrapper" between the data and the blackboard(s). Should this not be possible, consider creating a separate module cluster with its own blackboard, and recursively follow these guidelines.
f. **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.
g. **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, i.e. when the output of one is the input of another. Consider using two or more blackboards if (i) there is a wide range of information types in the total set of messages in the system, e.g. co-existing complex symbolic plan structures and simple boolean switches; (ii) there is a wide range of real-time requirements for the information flow, e.g. high-frequency vision processing and low-frequency plan announcements; (iii) 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.

4. **Test system against 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, etc., very early in the design process. Such end-to-end testing should also be performed regularly during development. The viability of the modularization is verified, with regard to the following:

a. **Expected communication (blackboard) throughput.** Network speed and computing power puts natural constraints on the maximum throughput of the blackboard. For example, perceptual data, that 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, etc. 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.
b. **Efficient information flow.** Static or semi-static 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 nature, as in visual processing and motor control. In such cases it is better to set up a server (hardware and/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.
c. **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 (i) the modules use the same set of data, (ii) are serially dependent on each other, (iii) have similar update frequency requirements, (iv) need to be tightly synchronized, or (v) are part of the same conceptual system (e.g. 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 (e.g. a user finishing a question) to the agent responding (e.g. 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 b and c above. 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 a few times over the design, 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.

5. **Iterate**. Go through steps 2-4 as often as needed.

6. **Assign 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 that is responsible for those modules operation, for defining the messages they receive and post. 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 their primary strength, and one Support, chosen by their interest and/or secondary strength. This step depends ultimately on the goals of the project and the experience/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.

7. **Test all 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 functionalities 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.

   This step often takes longer than expected. It is also one of the most overlooked steps, yet it that cannot be avoided – doing so will only force it to happen later in the process, delaying the ability to accurately estimate the project's full scope, and making it more likely that deadlines are missed and planned functionality has to be cancelled.

---

[6] http://freetts.sourceforge.net/docs/index.php
[7] http://java.sun.com/products/java-media/3D/
[8] http://www.extremeprogramming.org/

8. **Build the modules to their full specification**. This step is naturally done in frequent alteration 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 will dramatically decrease the amount of potential communication errors and make the system behavior more predictable. In addition to building the full modules, one must build extensive fine-tuning capabilities into each module, such as the timing of message posting, tracking of state and history, and to achieve the modules communicating as desired later on.

9. **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 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 and 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, which are reviewed and approved by the instructor. The proposals 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 whole system), the second phase focuses creating an integration plan, 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 their own personal/individual contribution to the project, identifying potential difficulties and challenges and proposing solutions to deal with them, as well as which team members they need to collaborate with. The description describes the module(s) and their functionalities that are to be coded, along with the input and output messages that the module(s) need, and which module(s) produce/consume them. The more detail in the messages' content and syntax in this write-up the better for the whole team, because it helps 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 2.

The fourth phase is the implementation. The leads for each part of the system, in collaboration with the instructor, help revise the project, reassign tasks, etc., as necessary as the projects progress. 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 the readers to evaluate its applicability in their own context.

To implement publish-subscribe functionality between modules we used Psyclone™ [CMLabs 2004], a platform which provides easy hookup via Java and C++ adapters. Psyclone shares some features with earlier efforts in this area, for example Cohen et al.'s Open Agent Architecture [1994] and Fink et al.'s DACS [1996]. 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 multi-blackboard 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 (InterSense IS900) 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. [9]

Our system has a detailed 3-D model of the entire room, including sofa, tables, chairs, desks, computers, etc., in which Mirage is placed, giving him a certain amount of knowledge about his surrounding. 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.

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

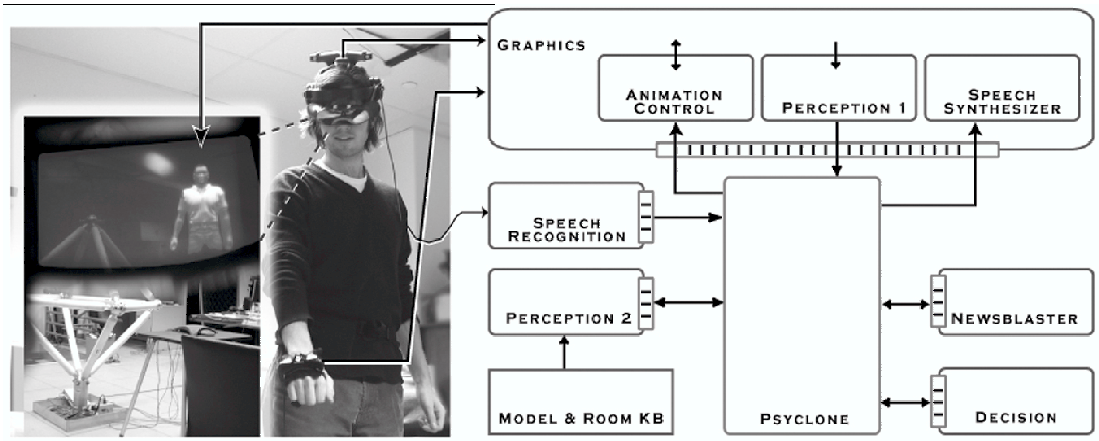

Figure 2. Mirage system architecture, showing sensor and display connections to the physical devices. Arrows to and from Psyclone indicate message and control flow. Small striped rectangles on the edges of modules signify Psyclone adapters.

Interaction Capabilities

We see future versions of Mirage as a tour guide in museums or tutoring employees about the layout of production plants, giving him potential as part of future commercial applications. 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 CMU[10]. 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 on-line 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 the user and greet.

Perception, Decision & 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 user's speech, proximity of the user to the Mirage agent, etc. Those messages are then used by the Decision module to create sensible responses to the user's input, and 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 agent's and user's position and orientation, noting changes of states such as "is the agent visible", "is the agent looking at the user", etc. Perception-1 uses technology based on previous work by Olwal, Benko and Feiner [Olwal et al. 2003] in which *SenseShapes* (volumetric regions of interest) are used to determine user's

[10] http://www.speech.cs.cmu.edu/sphinx/

selection and pointing. This relatively simple selection mechanism is triggered on any speech command, returning a list of objects being pointed at the time of speech. The output of Perception-1 is further processed by the module Perception-2, 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, etc., all interacting via message passing.

The main 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 perception, and forming a response action for the agent. That selected action is then sent to the Action Scheduling module that coordinates the avatar animation and speech synthesis. When the Decision module receives contradicting inputs or is unable to settle onto a single top-level decision, it can request that the avatar asks clarification questions, such as "Sorry, I did not get that. Please repeat".

Discussion

Two of the third-party components we used turned out to be inadequate for their function in our final system. Sphinx 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 actually generated more comprehensible output.

One potential problem of using existing modules is the sometimes lack of their control from the outside. Although such limitations turned out not to be a problem with Mirage (we had a clear idea up front 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", to be better suited for use with each other in multi-module, multi-author systems.

Psyclone was a good match to the Constructionist Methodology, and made it easy to distribute processes between machines, something that enabled us to quickly achieve acceptable run-time performance. This approach is likely to become increasingly

attractive over the coming years as computer prices keep dropping, making "garage-A.I." feasible and bringing new developers to the field with clusters of cheap yesteryear's computers, networks of which can supply the processing power for even serious A.I. work. The computer industry has no motivation to promote the re-use 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 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 where 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 timestamps 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 Modules	Speech
Subscribing Modules	Perception-1, Perception-2
Description	Speech stopped
Arguments	None

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

Message Type	Input.Perc.UniM.Voice.Speak
Sending Modules	Speech
Subscribing Modules	Perception-1, Perception-2
Description	Speech stopped
Arguments	*<sentence>* *<word>..</word>* *<word>..</word>* *</sentence>*

Table 2. Example specification of message format for modules.

Conclusion

Using the Constructionist Methodology 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 9 weeks, with 5 students dividing the work evenly, taking an estimated total of 2 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 A.I. majors – one had taken no introductory A.I. courses; the others had taken one. Students felt that the approach taken gave them 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 where most of the members are novices in A.I. This makes the Constructionist Methodology an excellent choice for classroom use.

Because the Constructionist Methodology was developed for integrating large numbers of modules and system hierarchies, it is not sensible to apply it to the re-invention 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 in 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, 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 2 mind-months strongly supports the hypothesis that the Constructionist Methodology speeds up system creation. A parallel can be found in the work by Maxwell et al. [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: 10 undergraduates working on the project for 8 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, having been used with good results in e.g. the design of a large virtual reality environment [Bryson & 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 (a) a novel, relatively broad, real-time system, developed by a small group of relatively inexperienced students working within a very short period of time; (b) team members' significantly heightened awareness of the functional breadth needed to construct interactive intelligent systems; (c) integration of prior work into a new system, and (d) 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 A.I. 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 the Constructionist Methodology could support. The approach applies 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 sub-systems, 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 A.I. and build a repository of "plug-and-play" A.I. modules and systems. We invite anyone with interest to join in this effort; the URL is MINDMAKERS.ORG.

Acknowledgments

The authors wish to thank Steven Feiner for access to his lab, 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 http://www.MINDMAKERS.org/projects/Psyclone. Psyclone is a trademark of Communicative Machines Laboratories. LEGO is a trademark of The LEGO Group, A/S.

References

Azuma, R. T. A Survey of Augmented Reality. In *Presence: Teleoperators and Virtual Environments*. Vol. 6 No. 4. August 1997. 355-385.

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

Bakker, B., & den Dulk, P. (1999). Causal Relationships and Relationships between Levels: The Modes of Description Perspective. *Proceedings of the Twenty-First Annual Conference of the Cognitive Science Society*, M. Hahn and S.C. Stoness (Eds.), 43-48.

Bischoff, R. (2000). Towards the Development of 'Plug-and-Play' Personal Robots. *1st IEEE-RAS International Conference on Humanoid Robots*. MIT, Cambridge, September 7-8.

Bischoff, R. & V. Graefe (1999). Integrating Vision, Touch and Natural Language in the Control of a Situation-Oriented Behavior-Based Humanoid Robot. *IEEE International Conference on Systems, Man, and Cybernetics,* pp. II-999 - II-1004. Tokyo, October.

Brooks, R. A. (1991). Intelligence without reason. *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, 569-595, Sydney, August.

Brooks, R.A., Ferrell, C., Irie, R., Kemp, C. C., Marjanovic, M., Scassellati, B., Williamson, M. (1998). Alternative Essences of Intelligence. *Proceedings of AAA1/IAAA*, 961-968.

Bryson, J. (2001). *Intelligence by design: Principles of modularity and coordination for engineering complex adaptive agents*. Ph.D. thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Bryson, J. (2000). Making Modularity Work: Combining Memory Systems and Intelligent Processes in a Dialog Agent. *AISB '00 Symposium on Designing a Functioning Mind*, 21-29.

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

CMLabs (2004). *The Psyclone Manual*. http://www.cmlabs.com/psyclone. Psyclone executables can be downloaded for Linux, Windows & Mac OS X from http://www.mindmakers.org/projects/psyclone.

Cohen P. R. , A. Cheyer, M. Wang, S. C. Baeg (1997). In M. N. Huhns and M. P. Singh (eds.), *Readings in Agents*, 197-204. San Francisco, CA: Morgan Kaufmann.

Cohen, Philip R., Adam J. Cheyer, Michelle Wang, and Soon Cheol Baeg (1994). An Open Agent Architecture. In Huhns, M.N. & Singh, M.P. (Eds.), *Readings in Agents*, 197-204. Morgan Kaufmann Publishers, San Francisco. Also available online at ftp://ftp.ai.sri.com/pub/papers/oaa-aaai94.ps.gz .

Laird, J. E. & van Lent, M. (2001). Human-Level AI's Killer Application: Interactive Computer Games. *AI Magazine*, summer, 15-25.

Laird, J. (2002). Research in Human-Level A.I. Using Computer Games.

Communications of the ACM, january, vol 45(1), 32-35.

Lucente, M. (2000). Conversational Interfaces for E-Commerce Applications. *Communications of the ACM (CACM), special issue on spoken interfaces*, September, 43(9), 59-61.

Fink, G. A., N. Jungclaus, F. Kummer, H. Ritter, G. Sagerer (1996). A Distributed System for Integrated Speech and Image Understanding. *International Symposium on Artificial Intelligence*, 117-126, Cancun, Mexico.

Fink, G. A., N. Jungclaus, H. Ritter, G. Saegerer (1995). A Communication Framework for Heterogeneous Distributed Pattern Analysis. *International Conference on Algorithms and Architectures for Parallel Processing*, 881-890, Brisbane, Australia.

FreeTTS. http://freetts.sourceforge.net/docs/index.php

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

Iba, S., C.J.J. Paredis, P. K. Khosla (2002). Interactive Multi-Modal Robot Programming. *Proceedings of ICRA,* Washington D.C., May.

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

Maes, P. (1990). Situated Agents can have Goals. In Maes, P. (editor), *Designing Autonomous Agents: Theory & Practice form Biology to Engineering and Back*, 49-70. MIT Press, Cambridge, MA.

Maes, P. (1990). *Designing Autonomous Agents: Theory & Practice form Biology to Engineering and Back*. Maes, P. (editor), MIT Press, Cambridge, MA.

Maxwell, B. A., L. A. Meeden, N. S. Addo, P. Dickson, N. Fairfield, N. Johnson, E. G. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter, & E. Silk (2001). REAPER: A Relfexive Architecture for Perceptive Agents. *A.I. Magazine*, spring, 53-66.

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

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

McKeown, K.R., R. Barzilay, D. Evans, V. Hatzivassiloglou J. L. Klavans, A. Nenkova, C. Sable, B. Schiffman, & S. Sigelman (2002). Tracking and Summarizing News on a Daily Basis with Columbia's Newsblaster. Human Language Technology '02 (*HLT* '02), San Diego, California.

McKevitt, P., S. Ó Nulláin, C. Mulvihill (2002) (eds.). *Language, Vision & Music*. Amsterdam: John Benjamins.

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

Newsblaster. http://www.cs.columbia.edu/nlp/newsblaster

Nicolesco, M. N. & M. J. Mataric (2002). A Hierarchical Architecture for Behavior-Based Robots. *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems Bologna*, Italy, July 15-19.

Olwal, A., Benko, H., Feiner, S (2003). SenseShapes: Using Statistical Geometry for Object Selection in a Multimodal Augmented Reality System. *Proceedings of The International Symposium on Mixed and Augmented Reality2003 (ISMAR*

03), October 7-10, 2003, Tokyo, Japan.

Rickel, J., J. Gratch, R. Hill, S. Marsella, W. Swartout (2001). Steve Goes to Bosnia: Towards a New Generation of Virtual Humans for Interactive Experiences, *Proceedings of AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*.

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

Thórisson, K. R. (2002). Natural Turn-Taking Needs No Manual: Computational Theory and Model, from Perception to Action. In B. Granström, D. House, I. Karlsson (Eds.), *Multimodality in Language and Speech Systems*, 173-207. 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. *First ACM International Conference on Autonomous Agents*, Mariott Hotel, Marina del Rey, California, February 5-8, 536-7

Thórisson, K. R. *Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills.* PhD Thesis, MIT Media Laboratory. 1996.

Terzopoulos, D. (1999). Artificial Life for Computer Graphics. Communications of the ACM, August, vol. 42(8), 33-42.

Kris Thórisson has been developing A.I. technologies for 15 years. As a co-director of the Wizard Group at LEGO Digital he developed a large virtual world inhabited by simulated interactive creatures. As VP of Engineering at Soliloquy Inc. he directed the development of natural language bots that assisted on-line customers of CNET and Buy.com. He has consulted on business and technology for numerous companies including British Telecom, Interactive Institute, Kaupthing Investment Bank, and NASA. He is currently CTO of Radar Networks Inc., a company he co-founded. Kris holds a Ph.D. from the M.I.T. Media Lab. He can be reached at kris@media.mit.edu.

Hrvoje Benko is currently a Ph.D. student working with Professor 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 will be starting his Ph.D. in machine learning at Carnegie Mellon University in Fall 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 2nd year 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 start-up companies. For the past few years he has been developing applications for the financial industry. His email is daa82@columbia.edu.

Aruchunan Vaseekaran is currently Vice President for R&D at m-Rateit.com, a startup firm that he co-founded. 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. Mr. Vaseekaran can be reached at vasee@ontash.net.