Co-Designing Agents*

Albert Goldfain and **Michael W. Kandefer** and **Stuart C. Shapiro**
Department of Computer Science and Engineering,
University at Buffalo, Buffalo, NY 14260
{ag33|mwk3|shapiro}@cse.buffalo.edu

Josephine Anstey
Department of Media Studies,
University at Buffalo, Buffalo, NY 14260
jranstey@buffalo.edu

Abstract

In this paper we sketch a new approach for agent design and operability called co-designing agents (CDA). As a working definition, we take a CDA to be an agent that participates in some aspect of its own design. The precise manner and degree of a CDA's participation will ultimately be a design-time decision, but by considering a specific agent implementation (that of AI actor-agents in a virtual drama) we are able to extract a general set of CDA requirements. The CDA approach utilizes concepts from several areas of active research in AI. We present a broad summary of the relevant literature and discuss its applicability to CDA design. We then consider the SNePS knowledge representation, reasoning, and acting system as a potential CDA implementation platform.

Introduction

The task or problem to which an AI agent is set impacts the entire design process, influencing the ultimate decisions on such things as agent architecture, implementation language, and knowledge representation scheme. Agent design techniques in AI are almost as varied as the applications in which the agents have been deployed. One interesting task which can be given to an AI agent is to participate in the design of AI agents. The lack of AI systems applied to design was brought up recently by Grosz (2005) and Nakakoji (2005):

> [I am] struck by how little has been done in developing one family of intelligent systems, 'intelligent project coaches' that participate with people in the design and operation of complex systems. As computer systems themselves become more complex, and their design and maintenance an ever more critical problem, building the capabilities for computer systems to participate intelligently in these endeavors seems even more important and the intellectual challenges of building such systems just as exciting (Grosz 2005)

> Although design as a discipline has been an enduring interest of both the AI and HCI communities, the impact of such interest has yet to be felt in the available design support tools. (Nakakoji 2005)

Both of these points suggest a need for systems that assist a designer *during* the design and testing phases of agent development. We will consider the implications of integrating a design-assisting system with the AI agent itself.

We define a Co-Designing Agent (CDA) as an agent that actively participates in some aspect of its own design. Both the human designer and AI agent are in the "design loop", cooperating and serving specific roles in the design task. In this paper, we flesh out a set of requirements that would allow a CDA to live up to its definition.

CDA and Cognitive Robotics

The CDA approach may be particularly useful for agents that can interact and affect the real world, either as fully embodied robots or agents with a simulated embodiment that interact with a human user. In this section, we consider how the CDA approach might impact robots with natural language competence and multi-agent robotic systems. We then turn to the domain of virtual drama, our original motivation for the CDA model, to extract a set of CDA requirements.

Robots with Natural Language Competence

If the agent is a natural language (NL) competent robot, that is, one that can both understand and generate NL utterances, it will be natural for a human user to ask it to do things that it does not yet know how to do, even though the task might be within its capabilities. An explanation amounts to a program that the robot should store for later use. This is a form of on-the-job learning.

For one example, Fevahr Cassie (Shapiro & Ismail 2003) knows how to go up, down, left, and right. We can tell her how to make a square and make a figure eight by building on that repertoire of actions, as shown here:

```
 : To make a square go up and right and down and left.
 I understand that to make a square go up and then go right
and then go down and then go left.

 : Make a square.
 I went up. I went right. I went down. I went left.
```

*Co-Designing Agents were the topic of a seminar at UB in the Spring 2005 semester, we would like to thank the other members of that seminar: Trupti Devdas Nayak, Carlos Lollett, Rahul Krishna, Shawna Matthews, and Rashmi Mudiyanur. We are also grateful to the members of the SNePS Research Group (SNeRG) for their comments on an earlier presentation of this work.

```
: To make a figure eight go up and left and down and right
  and then make a square.
I understand that to make a figure eight go up and then go left
and then go down and then go right and then make a square.

: Make a figure eight.
I went up. I went left. I went down. I went right.
I went up. I went right. I went down. I went left.
```

Using NL for on-the-job instruction has also been done by Lauria and colleagues in the context of a hardware robot that can be given directions for navigating in a small model city. Their robot participates in its design by using NL generation to ask for explanations of directions it doesn't understand:

User: Go to the library.
Robot: How do I go to the library?
User: Go to the post office, go straight ahead, the library is on your left (Lauria *et al.* 2001, p. 40).

Multiagent Robotic Systems

The CDA approach does not preclude systems of interacting individual robots. In a multiagent setting, the designer must address the needs of a community of cooperating agents as they work towards a common goal. Agents in a multiagent setting usually require a sophisticated method of coordination and planning to function properly.

The Robocup competition (see `http://www.robocup.org`) is a good example of a multiagent robotic setting involving soccer-playing robots. At a high level of development, one in which the base systems have been developed and the robots are learning to coordinate and form strategies, the designer takes on the role of coach and the robots take on the role of a team of players. In such a model, the players can report to the designer per their individual roles within the team (e.g., the role of a position during a certain play). The coach-designer and agent-player paradigm may be considered one instance of the CDA approach.

Virtual Drama Actor Agents

The CDA design approach was originally motivated by "The Trial The Trail", a virtual drama project (Shapiro *et al.* 2005; Nayak 2005; Anstey *et al.* 2004). The "actors" in the drama are automated SNePS-based AI agents in a virtual reality immersive environment (Anstey *et al.* 2003). The human participant interacts with the agents and the virtual world via: (1) a "wand", a hand-held participant-controlled device for navigation, and (2) a headset for tracking the participant's perspective. Like human actors, the agents are constrained by a predefined script. However, unlike typical scripts, there is room for an agent to respond to the human participant's actions. Such responses are not considered exceptions or error conditions, but are rather desirable improvisational behaviors meant to provide a more immersive and participant-specific experience. An agent's script is structured by a series of well-defined segments (acts and scenes), as well as a set of policies that handle participant-contingent behavior.

Agents are implemented using the GLAIR architecture and the SNePS knowledge representation, reasoning, and acting system (see the section on SNePS below). Each agent has access to several modalities of sensory input. IP sockets are used as a communication medium between the agent's embodiment in the virtual world and the agent's mind. A dedicated IP socket is assigned to each agent modality. An agent's embodiment in the virtual drama belongs to the agent alone. That is to say, agents must reason about the internal states of other agents and the human participant by utilizing their own (unique) point-of-view.

Using a traditional stimulus-response model for actor agents (in which each world event triggers a particular line of speech or agent reaction) can quickly lead to repetitive behavior by the agent. For instance, if an actor agent must respond to a user movement by saying "keep still", we would not want the agent to repeat this exact line for a rapid succession of user movements. Such unrealistic behavior is highly problematic in the domain of virtual drama. It exposes the actors as AI agents and gives the participant a less believable performance. This problem can be alleviated somewhat by providing more responses for the agent. However, there is no way for the designer to know *a priori* which scenes will evoke certain user behaviors. Even if the type of behavior might be anticipated, the duration, sequence, and mode of participant feedback will vary.

CDA Requirements

Problem domains such as virtual drama seem to demand a set of requirements that are not immediately addressed by simple stimulus-response systems (such as some production systems and reactive planners). These include:

- **Responsiveness to unexpected events**: The agent should be adaptable enough to react in a consistent and believable manner. This will keep the human participant on track through the plot.
- **Notion of a scene context**: The agent should be aware that a behavior (including speech, gesture, or act) may mean different things in different contexts.
- **Offline contingency planning**: The agent should anticipate situations that may lead to problems during a performance *before* the performance.
- **A model of itself**: The agent should have some description of its own embodiment and role in the drama.
- **Reporting ability**: The agent should produce reports of its actions after those actions are completed.
- **Explanation ability**: The agent should indicate, in some form of justification or explanation, the reasoning behind its choice of actions.

By designing agents with these additional abilities, the designer-agent relationship more closely resembles the human director-human actor relationship. The agent would take an active role in its own development.

We believe that the domain of virtual drama is sufficiently similar to other natural domains in which intelligent agents have been (and will be) deployed. For the remainder of the paper, we will consider how to best characterize the additional CDA responsibilities, and how various approaches from AI can benefit the CDA model.

The Job Cycle

The requirements for a CDA can be characterized as a cycle of necessary agent responsibilities. In this cycle, a three-part distinction is made with respect to the main "job" the agent must perform:

1. Before being "on the job":
   (a) Answer questions of the form "What would you do in situation x?"
   (b) Think of situations that may occur, but in which the CDA would not know what to do.
   (c) Discover contradictory beliefs.
   (d) Discover situations in which it is supposed to do contradictory actions.
   (e) Respond with possible plans for handling new requirements.
2. While "on the job":
   (a) Be able to answer questions of the form, "What are you doing, and why?"
   (b) Be able to answer questions of the form, "Why aren't you doing x?"
3. After being "on the job":
   (a) Be able to report what it did at various times.
   (b) Be able to report any problems it had, including situations it couldn't handle satisfactorily.
   (c) Be able to answer questions of the form, "Why did you do x?"
   (d) Be able to answer questions of the form, "Why didn't you do x?"

This is only a preliminary list of abilities and questions; indeed, several other interesting abilities could assist a co-design. We call this a job *cycle* because there may be successive iterations of the job. After job_1 may be before job_2. In each stage question-answering and reporting is crucial. Without feedback from the agent, the agent is not doing its part in the design process. Also, at each stage, the agent must be able to reason with models: (potentially) of itself, of the user, of the world, and of other agents.

A CDA's design goals are added to the task-specific goals of a traditional AI agent functioning in the relevant problem space. Thus, the CDA model will incur some overhead and should be used only when the benefit to the designer outweighs the cost.

Design Scenarios

Not all of the abilities listed in the previous section may be necessary for a particular task. Further fine tuning of requirements is imposed by various design scenarios.

The simplest model of distributing design responsibilities between an agent and its designer is one with a single designer working on a single agent. The designer in this case is the single trusted authority for the agent. Even in this simplest of cases, the designer may be conflicted regarding a design decision and could be assisted by a CDA in making those decisions.

A different design scenario would involve a single designer and multiple agents. This is the traditional model for multi-agent systems (MAS). Each agent may have a unique set of design goals as well as a unique set of task specific goals.

A third design scenario would involve multiple designers implementing a single agent. A particular agent design may be sufficiently complex to require several cooperating expert designers during implementation. Each expert might be assigned a particular module or subsystem of the agent. Unique avenues of participation open up for CDAs in such a situation. The CDA may be able to alert the designers when two design decision conflict across components. Such a scenario also brings up issues that the other scenarios do not, such as designer credibility (who should be trusted when design decisions are conflicting) and area of expertise (which designer would find a particular piece of information most useful).

Finally, a fourth design scenario would involve CDAs designing other CDAs. This is an ideal case in which the CDA is autonomous enough to apply co-design without any human intervention beyond the initial programming.

Related Work

The CDA model incorporates ideas from a number of active AI research areas. This section gives a general overview of several relevant systems that have been (or are currently being) developed. This overview is not exhaustive, and examines general AI topics as well as some specific agent implementations.

Planning Systems

Plans, in the AI literature, have been treated as "(possibly partial) representations of future behavior" (Tate 2001). Planning systems have been utilized in order to determine and constrain this behavior. These systems are essential for before-the-job and after-the-job activities. Before-the-job the agent can discuss its intended plans to complete on-the-job tasks. Meanwhile, the designer can correct or make suggestions regarding the plans. After-the-job, the agent can assess the overall success of its plans. The agent can describe the flaws in the plans it chose or suggest improvements to future plans.

The TRIPS system (Ferguson & Allen 1998) is prominently known for maintaining the human designer in the planning-system loop. In TRIPS, a human manager and the system work collaboratively to create plans in "crisis" situations. In this collaboration, the job of the system is to interpret the human user's utterances for constructing a plan, and acting upon them. Possible actions include: (1) providing information about the user-constructed plan(s), (2) modifying the plans based on new information, and (3) when requested, providing the results of alternative plans.

User-Modeling

The ability for an agent to model a designer is necessary in a multi-designer setting. In such settings, the agent requires knowledge of which designer is working on each of its components. Given a piece of information acquired on-the-job, the agent needs a method of determining the designer for which this information is most beneficial.

The field of user modeling has fostered a collaboration between the fields of AI and Human Computer Interactions (HCI). The goal of this collaboration is to create intelligent tools that learn to interact with an individual user's needs. The GOMS model (goals, operators, methods and selection rules) (John & Kieras 1996) is one user-modeling paradigm. GOMS is integrated with an existing piece of software that requires user modeling. The four GOMS properties are defined as follows:

1. **Goals**: the task of recognizing a user's goals and sub-goals while using the software
2. **Operators**: the operations available to the user through the software
3. **Methods**: learned plans composed of sub-goals and operators to accomplish a goal
4. **Selection Rules**: employed on a user-specific basis to decide on which method is most appropriate when multiple methods are available

User modeling is also relevant when a CDA's task requires it to deal with users that are not the designer. A designer may want to collect information about the different use cases or online user behaviors.

Multi-Agent Systems (MAS)

Much of the research done in MAS, in particular agent collaboration, can aid in the development of CDAs. Though many definitions have been given for MAS (Sycara 1998; Lesser 1995; Florez-Mandez 1999), a commonality is found in the following properties:

1. Each agent in the system is autonomous (to a designer-specified degree)
2. There is no global control of the system
3. Data is decentralized

MAS are similar to distributed artificial intelligence (DAI) (Bond & Gasser 1988), except that DAI includes some global control mechanism for the system. Like MAS, the CDA model does not presuppose a global control. The agent and designer act as separate, functioning agents and are aided through each other's capabilities and knowledge. Both agent and designer are a part of the agent's design process. Control of this system need not be entirely in the hands of the designer, and both can contribute to the design goals of the system. This collaboration is essential to all of the design scenarios given above.

Expert Systems

Expert systems have been employed for years as a means for delivering expert advice to naive users (Rheingold 2000). Though related to general question-answering systems, what distinguishes them from typical answer generating systems is user interaction. It is this component in implemented expert systems that can aid the CDA model.

Rheingold (2000) describes modern expert systems as having three components: (1) task-specific knowledge, (2) rules for making decisions, and (3) facility for answering questions about the system's own recommendations. The culmination of these three properties allows for a user to begin a learning interaction with the expert system. This interaction starts when the user engages a particular expert system; such as MYCIN, a medical diagnostic tool. Once the system is engaged a process is started in which the expert system requests information from the user in order to make an accurate assessment, and then provides a solution with an accuracy measure. This process is done using a combination of the first and second components. While an expert system can end at this point it does little to assuage any doubts a user might have with the results. It is here where question-answering regarding system recommendations is essential.

One system that supports this ability is Debrief (Johnson 1994). The Debrief system is implemented utilizing an episodic memory. This memory stores agent run-time actions alongside a condensed chunk of problem-state information related to these actions. After the run, a user of the system can ask the agent (through a menu interface) why an agent took a certain course of action. To answer the question, the agent first reconstructs the situation surrounding the user-selected action by retrieving the condensed information from episodic memory. The agent then checks the selected action's applicability, a test which involves "mental simulation" or "replay" of the recalled information. This test is done to ensure the Debrief recorded in episodic memory the situation prior to the action in question, and not the situation resulting from the action's execution. If the action is applicable, the agent attempts to determine the reason the selected action was chosen by continuing with this mental "replay". During this process information about this action "replay" is also stored, such as perceptions, stimuli, and conclusions. As a last step, this information is examined for those features that led to the action taken, as well as any alternative choices of action that were available to the agent.

Agent-based Project Management

Petrie (1999) describes a particular class of problems associated with design and development, in particular those dealing with distributed integrated project management. These problems involve changes in the design of an artifact and the subsequent effects on the development processes for the artifact, as well as a method of coordinating changes. The solution described involves the use of software agent nodes overseeing a particular aspect of the development process. The agent nodes receive messages about changes in the initial design or plan and assist in collaborating with other agent nodes to determine what has to change in the development process due to the design change.

Though the system described has its ties in project management, in particular those involving the construction of artifacts based on a plan, some of the aspects the system considers is related to the CDA model. The CDA model is based upon the idea that the agent will be continuously updated, and its design changed. Both agent and designer(s) contribute in this process, and must have an understanding of each other's roles in this process. The agent-based project management technique discussed could aid the agent in determining what design changes need to be heard by the de-

signer(s). This is not terribly useful in a single designer environment, as the designer would be the only source for design changes. However, in multiple designer scenarios, such a tool could help sift out what changes a particular designer needs to know about.

Domain Discovery Agents

A recurring feature of any design processes is the tendency for a piece of technology to be supplanted by a newer, more powerful version. In the realm of software, newer versions often include bugfixes to previous versions and enhanced capabilities. In many industrial software projects, a revision history is maintained to allow for backward-compatibility and traceability of changes.

For a CDA, it may be useful for the agent to have access to its own revision history. This would give the agent a log of design changes between job iterations. This information can be used by the agent to form a model of previous versions of itself and to reason about its previous abilities. Additionally, the agent could apply what was learned on the job towards building a future version of itself.

The Mars Rovers (`http://marsrovers.nasa.gov/home/`) are a good illustration of how human designers can benefit from allowing an agent to discover features of its environment and using this acquired information as feedback for future agent design. The Mars Pathfinder Sojourner Rover mission involved sending an unmanned craft to Mars to land and deploy a robotic agent capable of acquiring images. It is generally agreed that what was learned on this mission aided the designers in planning and engineering the newer, more powerful robotic geologist rovers Spirit and Opportunity (Squyres, Arvidson, & Bell 2004). The images taken by Sojourner, along with the experience of landing a rover on Mars suggested several engineering improvements. These improvements were applied in the design of the Spirit and Opportunity Rovers. Design improvements for the Spirit and Opportunity rovers were handled by human designers (partly) based on the online domain information collected by Sojourner. Thus, the ability to collect on-the-job information and to log this information as a revision history is crucial to many applications of CDA design. We also suggest that a CDA that has access to its own revision history would come closer to being capable of automating the inter-revision improvement process.

SNePS

Implementing a CDA requires a system capable of supporting a wide variety of features. The SNePS knowledge representation, reasoning and acting system (Shapiro & Rapaport 1987; 1995) is a good candidate for a CDA implementation platform. The underlying data structure of SNePS is a propositional semantic network representing the beliefs of a cognitive agent. Semantic networks are general enough to represent various kinds of information and, thus, can be flexibly extended to include features of the CDA model. The components of SNePS facilitate CDA development by allowing for many of the requirements described above.

SNIP, the SNePS inference package (Shapiro & The SNePS Implementation Group 2004), allows agents to infer new information from an existing set of beliefs. Inference is essential for CDA-designer communications, and for learning new information on the job.

SNeRE, the SNePS Rational Engine (Kumar 1993), is a model of acting for SNePS agents. SNeRE generates a trace of inference during action which can serve as an on-the-job report of its activities. SNePS agents can modify their underlying semantic network during an act, thus leaving an episodic memory of the act. SNeRE also allows SNePS agents to select plans based on context. Procedural information (i.e., "How do I do X?") is stored in the same semantic network as conceptual information (i.e., "What is X?").

SNeBR, the SNePS Belief Revision component (Shapiro & Johnson 2005), allows SNePS agents to detect contradictory beliefs during inference and report the inconsistency to the designer. The designer can then proceed to abandon one of the agent beliefs that led to the contradiction.

GLAIR, the Grounded Layered Architecture with Integrated Reasoning (Hexmoor & Shapiro 1997; Shapiro & Ismail 2003), is used for embodied SNePS agents (both physical and simulated embodiments). GLAIR specifies how an agent acquires beliefs about its own actions and percepts, and how it has a sense of its own situatedness in the world. This allows a SNePS-based CDA to obtain knowledge from on-the-job sensory experience.

SNePS agents are capable of metacognition (Shapiro *et al.* forthcoming), reasoning about their own reasoning and underlying representations. Metacognition is essential for supporting the self-modeling feature of a CDA. A CDA is an inherently incomplete (yet perpetually improving) agent and, as such, must be able to communicate its beliefs about itself to its designer.

For the purposes of communicating with the agent, SNePS provides two programatic interfaces for the designer. SNePSUL (a Lisp-like language) and SNePSLOG (a Prolog-like language). SNePS agents have also been provided a natural-language interface using an ATN parser/generator (Shapiro & The SNePS Implementation Group 2004).

Conclusion

We have described several requirements for the CDA model and some relevant literature from AI supporting this model. New developments in several fields we have not discussed (e.g., mixed-initiative planning and agent software engineering) will also be relevant to future development of the model. We have considered a job cycle characterization of CDA responsibilities. CDAs are a generalization of a specific type of agent which was motivated by the problem domain of virtual drama. Aspects of the CDA model have been added to the actor agent implementation in the form of an actor agent requesting more lines from the director when it has exhausted all of the ones it knows (Nayak 2005). This is a partial solution to the problem of repetetive participant behavior because it allows the agent to make its own determination as to whether a series of behaviors is repetitive enough to use up all relevant lines.

A designer should consider the following questions during the design phase of agent building:

- What design aspect can be automated by the agent?
- What are the benefits and costs of automating some design aspect?
- What shortcoming of a traditional agent design is overcome by using the CDA model?

By considering CDAs in different problem domains, the model can become a useful way to think about the role of an AI agent.

References

Anstey, J.; Pape, D.; Shapiro, S. C.; and Rao, V. 2003. Virtual Drama with Intelligent Agents. In Thwaites, H., ed., *Hybrid Reality: Art, Technology and the Human Factor, Proceedings of the Ninth International Conference on Virtual Systems and MultiMedia (VSMM)*. 521–528.

Anstey, J.; Pape, D.; Shapiro, S. C.; Telhan, O.; and Nayak, T. D. 2004. Psycho-Drama in VR. *Proceedings of The Fourth Conference on Computation Semiotics (COSIGN 2004)* 5–13.

Bond, A. H., and Gasser, L. 1988. An Analysis of Problems and Research in Distributed Artificial Intelligence. In Bond, A. H., and Gasser, L., eds., *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Publishers.

Ferguson, G., and Allen, J. 1998. TRIPS: An Intelligent Integrated Problem-Solving Assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. Menlo Park, CA: AAAI Press. 567–573.

Florez-Mandez, R. A. 1999. Towards a standardization of multi-agent system frameworks. *ACM Crossroads, Issue 5.4 on Intelligent Agents*.

Grosz, B. J. 2005. Whither AI: Identity Challenges of 1993-1995. *AI Magazine* 26(4):42–44.

Hexmoor, H., and Shapiro, S. C. 1997. Integrating Skill and Knowledge in Expert Agents. In Feltovich, P. J.; Ford, K. M.; and Hoffman, R. R., eds., *Expertise in Context*. Menlo Park, CA / Cambridge, MA: AAAI Press/MIT Press. 383–404.

John, B. E., and Kieras, D. E. 1996. Using GOMS for user interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction* 3(4):287–319.

Johnson, W. L. 1994. Agents that Learn to Explain Themselves. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)* 1257–1263.

Kumar, D. 1993. A unified model of acting and inference. *Twenty-Sixth Hawaii International Conference on System Sciences Volume III* 483–492.

Lauria, S.; Bugmann, G.; Kyriacou, T.; Bos, J.; and Klein, E. 2001. Training personal robots using natural language instruction. *IEEE Intelligent Systems* 16(5):38–45.

Lesser, V. 1995. Multiagent Systems: An Emerging Subdiscipline of AI. *ACM Computing Surveys* 27(3):342–343.

Nakakoji, K. 2005. Special Issue on Computational Approaches for Early Stages of Design. *Knowledge-Based Systems* 18:381–382.

Nayak, T. D. 2005. Patofil: An MGLAIR Agent for a Virtual Reality Drama. Technical Report SNeRG Technical Note 38, University at Buffalo.

Petrie, C.; Goldmann, S.; and Raquet, A. 1999. Agent-Based Project Management. *Lecture Notes in Computer Science* 1600:339–363.

Rheingold, H. 2000. Knowledge Engineers and Epistemological Entrepreneurs. In *Tools for Thought: The History and Future of Mind-Expanding Technology*. Cambridge, MA: MIT Press. 274–295.

Shapiro, S. C., and Ismail, H. O. 2003. Anchoring in a grounded layered architecture with integrated reasoning. *Robotics and Autonomous Systems* 43:97–108.

Shapiro, S. C., and Johnson, F. L. 2005. Automatic belief revision in SNePS. In Baral, C., and Truszczynski, M., eds., *Proceedings of the 8th International Workshop on Nonmonotonic Reasoning, Action, and Change*. Edinburgh, Scotland: IJCAI. 96–103.

Shapiro, S. C., and Rapaport, W. J. 1987. SNePS Considered as a Fully Intensional Propositional Semantic Network. In Cercone, N., and McCalla, G., eds., *The Knowledge Frontier: Essays in the Representation of Knowledge*. New York, NY: Springer Verlag. 262–315.

Shapiro, S. C., and Rapaport, W. J. 1995. An Introduction to a Computational Reader of Narrative. In Duchan, J. F.; Bruder, G. A.; and Hewitt, L. E., eds., *Deixis in Narrative: A Cognitive Science Perspective*. Hillsdale, NJ: Lawrence Erlbaum Associates. 79–105.

Shapiro, S. C., and The SNePS Implementation Group. 2004. SNePS 2.6.1 User's Manual.

Shapiro, S. C.; Anstey, J.; Pape, D. E.; Nayak, T. D.; Kandefer, M.; and Telhan, O. 2005. MGLAIR Agents in Virtual and other Graphical Environments. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)* 1704–1705.

Shapiro, S. C.; Rapaport, W. J.; Kandefer, M.; Johnson, F. L.; and Goldfain, A. forthcoming. Metacognition in SNePS. *AI Magazine*.

Squyres, S. W.; Arvidson, R. E.; and Bell, J. F. 2004. The Opportunity Rover's Athena Science Investigation at Meridiani Planum, Mars. *Science* 306(5702):1698–1703.

Sycara, K. 1998. Multiagent Systems. *AI Magazine* 19(2):79–92.

Tate, A. 2001. Planning. In Wilson, R. A., and Keil, F. C., eds., *The MIT Encyclopedia of the Cognitive Sciences*.