Design Issues for Mixed-Initiative Agent Systems

Thomas C. Hartrum and Scott A. DeLoach
Department of Electrical and Computer Engineering
Air Force Institute of Technology
2950 P Street
Wright-Patterson AFB, Ohio 45433-7765
{thomas.hartrum scott.deloach}@afit.af.mil

Abstract

This paper addresses the effect of mixed-initiative systems on multiagent systems design. A mixed-initiative system is one in which humans interact directly with software agents in a collaborative approach to problem solving. There are two main levels at which multiagent systems are designed: the domain level and the individual agent level. At the domain level, there are few unique challenges to mixed-initiative system design. However, at the individual agent level, the agent itself must be designed to interact with the human and the agent system, integrating the two into a single system.

Introduction

Much of the current research related to intelligent agents has focused on the capabilities and structure of individual agents. However, in order to solve complex problems, these agents must work cooperatively with other agents in a heterogeneous environment. This is the domain of *Multiagent Systems*. In multiagent systems, we are interested in the coordinated behavior of a system of individual agents to provide a system-level behavior. A unique type of multiagent system is one in which humans interact with software agents in a collaborative approach to problem solving. The goal of this paper is to describe some fundamental questions of how to design these mixed-initiative agent-based systems. We start by looking at the two fundamental levels at which design can take place – the domain level and the individual agent level.

Multiagent Systems Engineering

Our research in *Multiagent Systems Engineering* (MaSE) (DeLoach 1999) is an attempt to determine how to engineer practical multiagent systems. It uses the abstraction provided by multiagent systems for developing intelligent, distributed software systems. Our MaSE methodology is specifically designed for formal agent system synthesis. That is, we describe a multiagent system and its agents using formal languages and then transform the formal specifications into code.

In this research, we view MaSE as a further abstraction of the object-oriented paradigm where agents are at an even higher level of abstraction than typical objects. Instead of simple objects, with methods that can be invoked by other objects, agents coordinate their actions via conversations to accomplish individual and community goals. Interestingly, this viewpoint sidesteps the issues regarding what is or is not an *agent*. We view agents merely as a convenient abstraction, which may or may not possess intelligence. In this way, we handle intelligent and non-intelligent system components equally within the same framework.

The MaSE methodology is similar to traditional software engineering methodologies but specialized for use in the distributed agent paradigm. The methodology follows the basic steps shown in Figure 1. This methodology is somewhat different in that we design general components of our system before actually defining the system itself.

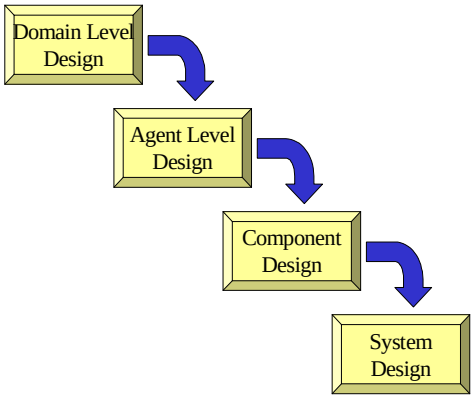

Figure 1. MaSE Methodology

Domain Level Design

The first step in MaSE is *domain level design*, which captures the basic types and interactions between agents in our system. By analyzing the complete domain, we get a bigger view of the overall system context. This context includes humans that interact with the system as well as other systems that might impact the overall operation of the system under development. At this level, whether or not an agent has intelligence, how that intelligence is captured, or how the agent is defined is not important. We are only concerned with the high-level definition of the types of agents, their goals, and their external interfaces. At the domain level, we see design as a problem of

modeling a distributed, collaborative system as a collection of heterogeneous agents that communicate with each other. The problem is partitioned so that each agent performs some task that contributes to solving the larger problem.

Once we have identified the types of agents, we identify possible interactions that might occur between different types of agents. These interactions become agent *conversations*, which are the specific types of communications that may occur between agents in the system. Conversations are defined using coordination protocols, which describe the possible sequences of messages that may be passed between agents to achieve coordination. Inter-agent communication is critical to the efficiency, effectiveness, and security of multiagent systems and can be defined using general purpose, or system specific, communication and security protocols.

Agent Level Design

The next step in MaSE is the *agent level design*. It is at this level that we define (or reuse) the agent architectures for each individual agent type. The agent architecture defines the components within each agent and how they interact. Here a formal specification of *how* the internal components of the agent work together defines the agent architecture while a specification of *what* each component should do defines the components. Together the specifications for the agent architecture and the components define the *behavior* of the individual agent. To integrate the agent into a multiagent system, each agent participates in a specific set of conversations. To complete the agent design, these conversations must be tied to specific information stored or produced within the agent itself. Once information required for specific conversations is tied to the information within the agent, the agent specification is complete.

Another agent-level issue is found in the case of agents that act as an interface to a specific resource such as a database or software tool. The formal specification for such an agent must frequently model interfacing the agent to an existing resource, such as a database or a software tool.

Component Design

The *component design* level is the next obvious level of the MaSE methodology. Once the agent architecture is defined, the components specified must be designed. Components being designed from scratch are defined using formal specifications. However, if components exist that can be re-used, it is our goal to define agents in such a way as to take advantage of existing component enabling technologies, such as JavaBeans, to allow us to reuse many components. Obvious agent components include planners, inference mechanisms, search algorithms, and learning algorithms.

System Design

Finally, *system design* takes place once the design of the domain, agents, and components are complete. By defining the domain first, system design becomes an exercise in picking the number and types of agents needed as well as defining specified parameters within the agent definition. Although not technically part of system design, once a system has been defined we can verify certain properties of interest such as safety and liveness.

Generating Multiagent Systems

Once the system has been completely specified from high-level system issues to low-level agent component behavior, the actual system can be generated. To formally generate multiagent systems, formal tools must first be used to specify the system. These formal specifications enable us to reuse existing components, synthesize new components, and analyze various properties of the system. The resulting system can then be tailored to a specific agent system *style*, such as used in JAFMAS (Chauhan 1997) where agents are generated in Java and use specific low level communications structures to carry out high-level agent conversations.

Formal Representation of Agents

Software synthesis based on formal specifications has been and continues to be a research area. Our current synthesis tool supporting agent systems is based on the use of Z specifications (Potter 1991) for defining hierarchically composed systems of objects, including both structural and behavioral aspects (Hartrum 1994). A simple example is the following specification for an inventory agent.

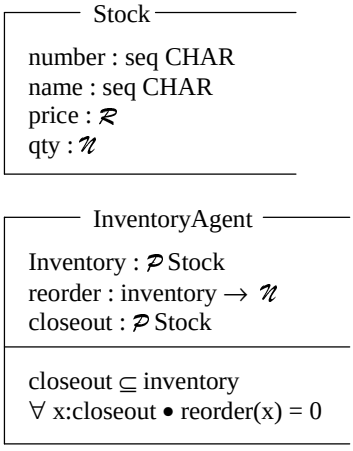

Figure 2. Structural Specification

The formal Z model is then parsed into our synthesis tool where reasoning software can verify various consistency

and correctness properties. The specification is then transformed into a formal design model from which object-oriented source code is generated.

The formal approach provides several benefits. It allows an abstract and mathematically precise way to specify the behavior of an agent system. It provides the mechanism for automated verification of system properties. Finally, it provides automated code generation.

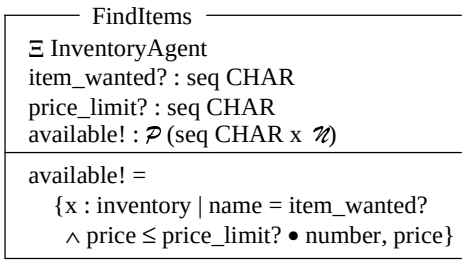

Figure 3. Functional Specification

State	Receiver	Next State	Action	Send
Idle	request(item, limit)	Searching		acknowledge
Searching		Idle	FindItems(item, limit, available)	Result(available)

Figure 4. State Transition Table

The Z *schema* named Stock, Figure 2, is a formal specification for an object class representing a single stock item, with attributes representing its stock number, item name, price, and quantity on hand. Similarly the schema InventoryAgent is an aggregate representation of an inventory system with a set of stock items (inventory), a set of discontinued items (closeout), and a total function mapping each stock item to its reorder level. The lower part of the schema represents invariant constraints, in this case that the closeout items are included in the inventory set, and that the reorder point for all closeout items is zero. The agent can perform a task, specified in Figure 3, of finding the set of stock items with a specified name whose price are below a certain limit. The output of this task (essentially a function, or process) is a set of item number and price pairs. Finally, the dynamic behavior specified in the state transition table indicates that the agent waits in the Idle state until it receives a request for inventory information. After sending an acknowledgement to the requestor it enters the Searching state and invokes the FindItem task. Upon finishing, it sends the result to the requestor and resumes the Idle state.

Mixed-Initiative Issues

A mixed-initiative agent system is one in which some of the agents are pure software, frequently interfaced to a database or software tool resource, and some of the agents are effectively human in-the-loop experts. *Human/agents*, which we define as software agents interfaced to a human as their local "resource," represent the latter. As implied by the name, a human/agent is really a team consisting of a human and an interface agent, which allows the human to interface to the rest of the system. We see three models of interaction between these human/agents and the associated multiagent system.

- *Client-server with the human as client*. In this model the human basically inputs a *query* to its interface agent, who, after coordinating with other agents in the system as needed, provides a *response* back to the human. This process could be a simple query-answer session or could involve asking the human for clarification or additional information. A variation of this role is when the human inputs a *task* to the agent to be performed, such as "activate appropriate alarms.'' The overall response to the human may simply be acknowledgement of accomplishment (or failure) of the task, or might result in some information being returned, similar to a query. Again the human might be asked for additional information, clarification, or suggestions on how to re-accomplish a failed task.
- *Client-server with the human as server*. In this model, an online human might receive a query for information from an agent acting on behalf of another agent or human. For example the agent might suddenly pop up and inquire about the availability of a time block for a meeting. In this case the human would provide a response that would be returned to the original requestor. In this model, the human may, or may not, receive any formal feedback about the results given by the answer provided. In addition, the human may need to query the sender for more information for clarification or even justification of the original request.
- *Peer-to-Peer*. This model is more in the spirit of a true collaborative approach. In this model a group of software agents and human/agents cooperate to solve a problem. The human participates in both client and server roles at different times during the session. Because this method of interaction encompasses the first two, we will concentrate on it for the remainder of the paper.

The issues of interest here are those unique to the mixed-initiative concept. In the context of the two levels at which multi-agent systems can be considered, the primary issues for mixed-initiative systems seem to be at the individual agent level. At the multiagent domain level, there are no unique challenges since the interface agent part of the human/agent will encapsulate the human and will look identical to the software agents that comprise the rest of the system. The only potential difference at the domain level might be in terms of the response time. Whereas a software agent will typically process messages fairly quickly, the human part of the human/agent will usually be significantly slower to respond. Even if the human

responds immediately, the total response time will typically be seconds and may be minutes or hours, depending on the length of the average coffee break.

At the individual agent level we see two primary factors that affect the design of human/agent as compared to a software agent. The first factor concerns the uniqueness of the data structures and behaviors of a human/agent as opposed to other software agents, and how this might affect specific design issues. The second consideration deals with the special case of interfacing to a "human" resource. Clearly this could build on existing research into *user-interface-agents*, wherein the agent learns the user's preferences and tailors the interface to that specific user. However, a more direct concern involves defining the most appropriate form of the interface (e.g. a table *vs.* a diagram) based on matching a formal description of the problem to formal specifications of the various representation forms.

Human/agent Considerations

Since we see no significant issues regarding human/agents at the domain level, we will address the rest of the paper to the issues related to the individual human/agent design. The first issue involves modeling the underlying architecture and behavior of such a specialized agent. In a human/agent the architecture needs to support both the client and the server roles. Thus it needs to respond to *ad hoc* inputs from the human as well as to asynchronous inputs from other software agents. It must also handle human responses to queries from other agents, as well as send queries on behalf of the human. Clearly each individual human/agent would have to be tailored to both the particular problem being solved and to the individual expertise of the human to which it must interface. However, it would seem that the common parts could be captured as a reusable architecture or *set* of architectures on which an agent designer could build a particular agent.

The second issue involves the human-computer interface through which the human/agent communicates with the human. If all agent collaborators were human/agents, communication via text frames (e.g., e-mail) would be appropriate. However, in a mixed-initiative system, data must be exchanged readily between human/agents and fully automated agents. Thus the interface to the human needs to support more formal data structures, including tables, enumerated values, binary decisions, graphs and other graphical representations, etc.

A Proposed Human/agent Model

We propose the following human/agent model. The human/agent is basically a concurrent state machine, where each state model represents a conversation. That is, the human/agent can concurrently process multiple simultaneous conversations. On the one hand, the human/agent could receive multiple requests, overlapping in time. Thus there would be multiple open tasks that the human is working on at any given time. On the other hand, in order to process a single request, the human might need more information, and would send several requests for needed information to the system. Thus there could also be several pending requests for information at any one time.

Within our model there are two types of human/agent conversations. The first is *transaction-based*. In this model, a single query comes to the human/agent and is presented to the human. The human agent processes it, possibly sending out one or more transaction-based requests and waiting for their response before finishing the task. When done, the human responds to the original query in a single, final response by providing the necessary information and invoking some form of *submit*, which the human/agent then sends back to the requestor in a single response message.

The other type of conversation in our model is *incremental-based*. In this model several agents, including the human/agent, share data that is incrementally updated. The human/agent displays the shared information on the screen, and dynamically updates it as other agents submit incremental changes. Similar to a traditional blackboard system, all agents effectively write information onto a common blackboard, which is displayed for the human to monitor at all times. For example, two university departments may be collaborating to schedule courses in a non-conflicting way. The "blackboard" would display the current schedule, and both parties could independently add courses to the schedule. A variant of this model would allow each agent to register for the information it was interested in, and only that information would be sent to it. Continuing the previous example, another agent might be concerned with scheduling classrooms. It would also need to see the results of the two course-scheduling agents, and for each course scheduled might need to access another agent to determine course enrollment. However, neither of the first two agents would be concerned with either room information or enrollment information, and would simply not register for that information.

Human/agent Design Issues

There are several design issues related to the human/agent model described in the preceding section. While the model is intended to represent any human/agent, clearly there will be differences in the agents themselves depending primarily on the type of problem support being provided by the human and the particular preferences of the human himself. Some of these are defined below.

1. *What are the allowable queries to be sent to the agent?* This basically defines the agent's interface as seen by the rest of the system. The allowable

queries are obviously dependent on the types of problems that the agent is designed to handle.

2. *What input information needs to be supplied with each query?* Based on the answer to question 1, this defines the information needed to support each query.
3. *What is the syntax of the query messages?* Based on questions 1 and 2, this ultimately defines the syntax of the agent interface, as seen by the other agents.
4. *What information is needed to answer the query?* For each query, this defines the information that makes up the required answer.
5. *What is the syntax of the response messages?* Based on question 4, this also defines the syntax of the agent interface, as seen by the other agents.
6. *For each query, what is the appropriate form of information exchange with the human?* This defines the appropriate form (or user-selectable forms) in which to present the query and its information to the human, and the form in which the human will provide the response information.
7. *For an incremental based approach, what information needs to be presented, how should it be presented, and what is the syntax of the corresponding update messages?* Basically questions 1 through 6 need to be re-interpreted in the context of the incremental-based approach.

It is interesting that the answers to these questions relate directly the design of the human interface to the human/agent as well as the system interface to the human/agent. If generalized, the questions are also applicable to software agents as well as human/agents.

It is our hypothesis that these design decisions can be supported by a formally based design tool that would aid the software engineer (agent designer) in specifying a specific human/agent for a particular project in such a way that the resulting agent code could be automatically generated.

References

DeLoach, S. A. 1999, *Multiagent Systems Engineering A Methodology and Language for Designing Agent Systems*. to appear in Workshop Notes of Agent Oriented Information Systems (AOIS '99), Seattle WA, 1999.

Chauhan, D. 1997, *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*, Thesis, ECECS Department, University of Cincinnati.

Hartrum, T. C. and P. D. Bailor 1994, *Teaching Formal Extensions of Informal-Based Object-Oriented Analysis Methodologies*, 7th SEI CSEE Conference, San Antonio, TX, Jan. 1994.

Potter, B. et at 1991, *An Introduction to Formal Specification and Z*, Prentice Hall, 1991.