Minutes from "Intelligent Agents" in *Multiagent Systems* [2]

Robert Glaubius
Lecture by Tibor Moldovan and Shabbir Syed
CSCE 976 Advanced Artificial Intelligence

April 01, 2002

1 Introduction

In Monday's lecture, we covered material from the first chapter of [2], entitled "Intelligent Agents" and authored by Michael Wooldridge. This chapter covers the basis of agent architecture, with a look at several different concrete architectures, and some programming languages that focus on the use of agents.

This material was presented by Tibor Moldovan and Shabbir Syed. Tibor covered the first portion of the chapter, which included an introduction to agents, the general architecture of agents, as well as the first concrete architecture. Shabbir covered the latter three architectures, as well as the agent programming languages.

2 Tibor's Lecture

2.1 What is an Agent?

Tibor began his portion of the lecture by discussing the attributes of an agent[1]. We reviewed that an agent is a computer system situated in an environment that is capable of autonomous action in order to meet its design objectives. Usually an agent has some set of sensors that enables it to perceive its environment [2]. Some classical examples of agents are thermostats, cruise control, and software daemons.

There are five different properties used to describe an agent's environment; these directly effect the complexity of tasks performed by an agent. These properties are as follows, with explanations drawn from [2], pgs. 30-31.

Accessible vs. Inaccessible: The accessibility of an environment is the degree to which it has access to the current environment state. An accessible environment implies an omnipotent agent.

Deterministic vs. Non-deterministic: In a deterministic environment, each action leads to exactly one consequent state. Non-determinism indicates that an action may result in one state out of some set of possible outcomes.

Episodic vs. Non-episodic: In an episodic environment, agent performance is dependent upon a number of discrete episodes, while in a non-episodic environment, performance is dependent upon the sum of previous experience.

[1] Unfortunately, Tibor neglected to elucidate Hugo Weaving's relation to the agent framework, so perhaps viewing "The Matrix" as a class will be necessary to fully comprehend this point.

[2] It is important to mention at this point that this definition is controversial, and that many definitions of agency exist in the literature

Static vs. Dynamic: A static environment is one in which the only changes in environment state are due to the agent's actions. In a dynamic environment, there may be many other factors that alter the environment state.

Discrete vs. Continuous: An environment is discrete if there is a limit to the number of possible actions and precepts, while in a continuous environment no such limit exists.

More information on these properties can be obtained from [1]. Tibor noted that the most complex environment is inaccessible, non-deterministic, non-episodic, dynamic, and continuous. In other words, an environment that has the properties of the natural environment.

The next feature of agents that we discussed after environment was autonomy. Autonomy implies that the agent is in charge of what action it will perform. This is distinct from the idea of a software object, which has methods that outside agencies can use to control the object. The catch-phrase used to characterize the relationship between objects and agents is that *Objects do it for free, agents do it for money*. Another point mentioned by Tibor was the issue of control; agents have their own thread of control, while objects do not.

The discussion of agents at this point led into the topic of intelligent agents. The distinction between an agent that is intelligent and one that is not is that an intelligent agent is capable of flexible behavior. Flexibility is defined as reactivity, pro-activeness, and social ability. Reactivity is the ability to perceive and respond in a timely fashion, pro-activeness indicates the ability to take the initiative in pursuit of goals, and social ability is the ability to interact and communicate. The idea of social ability is especially relevant in the domain of multiagent systems.

2.2 Abstract Architectures for Intelligent Agents

The next topic in Tibor's lecture was the abstract design of intelligent agents. The design of an intelligent agent involves the definition of the environment as a set $S = \{s_1, s_2, \ldots\}$ of all possible environment states, the effectoric capacity of the agent as the set $A = \{a_1, a_2, \ldots\}$ of actions, and views the agent as a function $S^* \rightarrow A$, which maps sequences of environment states to actions.

The interaction of an agent with its environment can be represented as a history. The *history* is a sequence of environment states with transitions annotated by the action that caused the transition, as shown below.

$h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots s_u \xrightarrow{a_u} \cdots$

We are generally interested in agents whose interaction with the environment does not end; therefore their histories will be infinite.

Based on the use of histories, we can categorize intelligent agents into two abstract classes. These are purely reactive agents, and agents that maintain a state. The purely reactive agent[3] is one whose action function maps a single environment state to an action, as shown in Equation 1. This implies that the agent is reasoning only from the current environment state. Dr. Choueiry noted that an alternative function would be $S \times h \rightarrow A$, which maps a state and history to an action.

$$Action : S \rightarrow A \tag{1}$$

Alternatively, an agent with state maintains an internal state that maintains history and environment information. The set of these internal states is denoted I. An example agent of this type could use the function $See : S \rightarrow P$ to map environment states to percepts. Based on this function, we can then define a transition function next $Next : I \times P \rightarrow I$ that maps an internal state and percept to an internal state. This

[3]This type of agent is called a *simple reflex agent* in [1].

is then used to define the action function $Action : I \rightarrow A$, mapping the internal state to an action. This concept of an internal state will allow this type of agent to perform a superset of the tasks that a purely reactive agent can perform.

2.3 Concrete Architectures, Part I

Tibor then moved on to the next portion of the paper, discussing the first concrete architecture reviewed in [3]. This is the logic-based agent.

A logic-based agent is an agent that relies on the rules of first-order logic to represent the environment and decide on actions. Percepts are represented as true or false statements in the set of environment states S. Let L is the set of all FOL sentences, and D is the power set of L, then internal state of a logic-based agent is an element of D. A logic-based agent makes decisions using a set of deduction rules. In a sample logic-based agent, we could have the function $See : S \rightarrow P$ as before, and the function $Next : D \times P \rightarrow D$ that maps a database and a precept to a new database. The action rule $Action : D \rightarrow A$ then maps a database to an action.

Tibor describes reasoning in a logic-based agent is as follows. agent tries to find an action that satisfies some desired internal state (e.g. its goal state). If such an action exists, that action is taken and evaluated as TRUE. Otherwise, the agent tries to find some action that is consistent with the goal state. If such an action exists, it is taken, otherwise a no-op is performed, as no action will achieve the desired state. The agent's performance is based on its deduction rules and its current database.

The advantages of logic-based agency are primarily due to the elegance of formal logic. Further, as long as the environment doesn't change, and FOL inference is carried out correctly, the agent will find an optimal action if one exists. The problem is that logical inference is a hard problem; therefore it is difficult to guarantee that the environment will remain stable while reasoning is carried out. This property is called *Calculative rationality*.

The primary disadvantages of logic-based agency are threefold. First, it is difficult to map percepts into logic statements. The next is the difficulty in representation of dynamic environments. Temporal reasoning tends to be slow, which leads to cases in which the optimal decision is obsolete before it is even discovered. Finally, procedural knowledge is not necessarily intuitive once represented in logic.

3 Shabbir's Lecture

3.1 Concrete Architectures, Part II

Shabbir's portion of the lecture moved on from Logic-based agents to the next architecture described in the paper. This was the reactive architecture, as exemplified by Rodney Brooks's Subsumption architecture.

This architecture doesn't use an intermediate representation for describing environment states, but instead maps directly between percept and action. The subsumption architecture is named for its mechanism for prioritizing actions. A given environment state can cause several actions to "fire", e.g. be selected for execution. Active actions subsume each other, as the action with highest priority is executed. To facilitate this, the possible behaviors must be totally ordered. The subsumption architecture is suggested by the emergent behavior paradigm. This is the idea that the interaction between a set of simple behaviors can achieve complex results.

Shabbir discussed the application of the subsumption architecture given in the paper. This is the example of the rock collecting agent problem in the paper. This example demonstrates the importance of proper ordering for the behaviors, as well as providing a good example of how complex interaction can arise from simple behavior.

In summary, we have a number of agents that are required to find and collect clustered samples of some rock. The landscape renders the agents incapable of direct communication. We are also given a base that emits a signal that degrades with distance. Each agent has several behaviors to choose from, including the following.

1.6 If an obstacle is detected then change direction.

1.7 If carrying samples and at base then drop samples.

1.8 If carrying samples and not at base then travel up signal gradient.

1.9 If a sample is detected then collect sample

1.10 If TRUE then move randomly

These behaviors are ordered $(1.6) \prec (1.7) \prec (1.8) \prec (1.9) \prec (1.10)$, where we would choose the leftmost of all firing behaviors. These behaviors will enable the agents to collect samples, but won't offer much in the way of cooperation. For this reason, we add the following behaviors.

1.12 If carrying samples and not at base then drop 2 crumbs and travel up gradient.

1.13 If crumbs detected then pick up 1 crumb and travel down gradient.

Given these behaviors[4] , we modify the ordering to $(1.6) \prec (1.7) \prec (1.12) \prec (1.9) \prec (1.13) \prec (1.10)$. Also notice that now the agents will establish a form of indirect communication. Once a sample cluster is found, a trail from it to the base will be created. Other agents will find this trail and commence collection of the same cluster; until the cluster is exhausted, the trail will be strengthened. Once the cluster is exhausted, radioactive decay (as noted by Dan) and wandering agents will weaken the trail. As Dr. Choueiry noted, this is inspired by the apparent behavior of ant colonies.

The advantages of the subsumption architecture are its simplicity and tractability. Another strength is its robustness; the removal of a single agent doesn't substantially detract from the completion of the task. Its disadvantages are that decision making is local, and its unclear how to include global knowledge into the computation. Another weakness is the absence of learning from experience. A final difficulty is that it is not necessarily intuitive to judge how a set of simple behaviors and environment will interact, and design is frequently based on trial and error.

Next, Shabbir moved on to describe the Belief-Desire-Intention (BDI) architecture. These architectures are based in the philosophical tradition of understanding practical reasoning. It focuses on deliberation about intentions, e.g. what goals do we wish to achieve. It employs means-end reasoning, in other words, given the conditions, how are we going to achieve these goals. Dr. Choueiry enforced how this is can be represented as a series of if-then-else rules, and explained that the foundation of means-end reasoning in computer science stems from work by McCarthy and Newell during the 1960's.

An important aspect of BDI architectures is the role of intentions. Intentions serve four important roles. The drive means-end reasoning and constrain future deliberation by focusing on a goal set. They persist through time until they are accomplished or become impractical, and influence beliefs that direct future practical reasoning.

BDI architectures must make a tradeoff between the degree of commitment and reconsideration. The extremes are bold and cautious agents, where a bold agent is one that makes strong commitments and rarely reconsiders, while the converse is true of a cautious agent. Let γ be a parameter that characterizes

[4]Scribe's note: behavior 1.11 omitted, as it is the same as behavior 1.7.

the rate of change of the world. If γ is low, indicating that the world changes slowly, then bold agents will outperform cautious agents. If γ is high, then cautious agents will outperform bold agents.

There was some contention about this; Dan wondered if the bold agent wouldn't still outperform the cautious agent when γ was high simply because the cautious agent would never actually do anything while the bold did. Praveen explained that, since the bold agent may commit to acts that are detrimental in the current environment state.

The BDI architecture requires that the following functions are defined. $Options : \wp(Bel)^* \times \wp(Int) \rightarrow \wp(Des)$ maps beliefs and intentions to desires. $Filter : \wp(Bel)^* \times \wp(Int) * \times \wp(Des) \rightarrow \wp(Int)$ filters out intentions that aren't compatible with the beliefs and desires of the agent. $Execute : \wp(Int) \rightarrow A$ maps intentions to actions. Using these, we can then define the agent decision function $action : P \rightarrow A$. This function sets B to be the set of beliefs consistent with the current percepts, then sets D to be the result of the $Options$ function given B. I is then set to be the result of $Filter$ given B, D, and the old set of intentions I. We then call $execute(I)$.

After the discussion of BDI architectures, Shabbir went on to discuss layered architectures. These architectures utilize several different agents and architectures to respond to different situations. Two layering strategies were discussed - the horizontal and vertical layerings. We discussed an example of each; these are the Touring machine, which uses a horizontal layering, and InteRRap, which uses a two pass vertical layering strategy.

In a horizontal layering strategy, percepts are passed directly to several layers. Each layer is a distinct agent architecture, and recommends an action. A control subsystem then chooses one of these actions to execute. The primary issue in such an agent then is determining which layer has control of a particular decision.

In a vertical layering strategy, there are two potential control strategies. These are the one and two pass control strategies. In a one pass control strategy, percepts and information travel upwards through the layers. In the two pass control strategy, information travels upwards until it reaches a layer that is competent to recommend an action. This layer can then delegate subtasks downwards to lower agents until an action decision is made.

The Innes Ferguson's Touring machine is an example of a horizontal layering system. It is comprised of three layers; a reactive layer, a planning layer, and a modeling layer. The reactive layer essentially behaves like a purely reactive agent as discussed earlier, and offers immediate response to an input percept. The planning layer employs a set of skeletal plans to derive courses of action. The modeling layer represents entities in the world, and predicts conflict between agents and generates goals to resolve these conflicts. The Touring machine employs a mechanism for control that inhibits the passing of percepts to agents that are known incompetent in dealing with such input.

The InteRRap system, designed by Joerg Mueller employs a two pass vertical layering. It uses three layers, a behavior layer, a plan layer, and a cooperation layer. These are analogous to the reactive, planning, and modeling layers from the discussion of Touring machines. A world interface sends information first to the behavior layer, which has access to a world model. The behavior layer can pass information upwards to the planning layer, which has access to a database of planning knowledge. This layer in turn can communicate upwards with the cooperation layer, which has a KB of social knowledge. Each layer in the InteRRap system has several properties. The first is situation recognition, which maps a KB and current goals to a new set of goals. Goal activation selects a plan to execute based on the current plans, goals, and KB.

The primary differences between the Touring Machine and InteRRap (aside from the layering system) are as follows. InteRRap uses a series of knowledge bases, while the Touring machine does not. In the Touring machine, layers are able to directly receive input and generate output, necessitating the use of a control layer. InteRRap relies on interaction between the layers to govern the system.

Layered architectures are more common, since the layering allows for decomposition of functionality.

Simply put, we can use a layer that specializes in certain tasks to handle a subset of the tasks required of the layered agent. Layered architectures do lack the conceptual and semantic clarity of the unlayered architectures.

3.2 Agent Programming Languages

We then went on to briefly discuss the agent programming languages mentioned in the paper. These were Agent0 and Concurrent METATEM. Agent0 is a prototype language developed by Yoav Shoham in 1990, and embodies agent-oriented programming. Concurrent METATEM is a more practical language that uses logic formulae, and was developed by Michael Fisher in 1994.

The components of Agent0 are primitives for expressing the high-level concepts necessary for describing agents. These are mechanisms for specifying a set of initial capabilities, beliefs, intentions, and commitment rules. A commitment rule has a message condition, a mental condition, and an action. Such a rule fires when the message condition matches against messages the agent has received and the mental condition matches against the agent's beliefs. This causes the action specified by the rule to fire. These actions may be private or communicative to other agents. Dr. Choueiry noted that this extends rule-based systems to include a notion of belief and intention.

Concurrent METATEM specifies agents using temporal logic. This specification can be executed directly in order to generate the agent's behavior. A conjunction of $P_i \Rightarrow F_i$ rules are used to match against an internal history; if a match is found, the rule fires. When a rule fires, the commitment is updated to include the consequent of the rule. METATEM is concurrent in that agents communicate asynchronously.

Shabbir then concluded with a summary of the discussion, in which we mentioned in the paper. These were Agent0 and Concurrent METATEM covered the definition of an agent, and the importance of agents. The goal of this research area is the theory, design, construction and implementation of intelligent agents.

4 Questions

Amy

- wondered what the agent-based languages were being used for. Tibor responded that they were useful for providing predefined primitives. Shabbir also noted that some systems have been implemented using these languages. Choueiry added that the focus of the paper was not on application, as well as reaffirming Tibor's point that the languages also provide communication protocols for inter-agent communication.
- Noted that a good job was done of integrating examples of real systems into the paper
- Also mentioned in her comments that abstraction of state information to reduce the number of potential states is a useful area. She notes that this would lose information, however.

Cory

- "Ah, yes, I remember agents from 476"
- Mentioned that there are similarities between the idea of agents and a task Cory, Dan, and Brian had performed at the ACM contest.
- Notes that it would take a large number of threads for a good implementation of a system that requires many agents.

Praveen

- Points out that the line "Objects do it for free, agents do it for money" explains the difference between agents and objects beautifully.
- Wondered what it means when we compile the logical specification of an agent. *Scribe's reply: If I remember correctly, this would be regarding InteRRap, which uses a logical syntax. We*

could therefore compile our agent in roughly the same sense that we compile a block of code in any other non-interpreted language.

item [Lin Xu]

- Points out that this chapter is a deeper discussion of the same basic ideas from [1]. Some of the definitions are different, however, such as that of 'agent with state'.
- Asks that, since the logic-based approach is at a disadvantage for representing the real world, which approach does this better? *Scribe's reply: I suppose that the most natural answer is to point out that reactive agents avoid this problem by ignoring an intermediate representation altogether. If your input is an image of the real world, then we map an action from the pixels. The problem here is that it is often difficult to determine the pattern of reasoning that leads to a decision for sufficiently complex input.*

Dan

- He wonders why anyone would confuse objects and agents in the first place. It seems that the section that tackles this issue is largely irrelevant and could be abbreviated greatly. *Scribe's reply: Since I tend to agree with this point, there's not much I can say. I suppose it's an artifact of the focus on object-oriented paradigm that some of the features of agents are comparable to some of the high-level aspects of objects.*
- Extolls the virtues of the 'robots exploring a foreign body with communication' example.
- Expresses the sentiment that the agent-based programming languages look "awful". Believes that a good class hierarchy should suffice in place of such tools.

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence, a Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[2] Gerhard Weiss, editor. *Multiagent Systems*. MIT Press, 1999.

[3] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems*. MIT Press, 1999.