Incorporating a Runtime Interactive Visualization into Intelligent Agents

Aizhong Lin and Mao Lin Huang
Faculty of Information Technology,
University of Technology, Sydney
POBox 123, Broadway
NSW 2007, AUSTRALIA

Abstract

Two reasons motivate us to incorporate a real-time interactive visualization into intelligent agents. Firstly a visual presentation gives users a better understanding of the runtime works that an agent is currently doing, so that they can evaluate the performance of an agent through the visual user interface. Furthermore, a real-time interactive visualization provides functions to view the inner state transitions of an intelligent agent. Secondly a real-time interactive visualization can support direct user interference while an agent is running. For example, the user of an intelligent agent may want to interrupt the running of the agent to rectify its process if they feel the plan chosen by the agent is not good enough to achieve a goal or they believe an agent could fail to achieve a goal. This paper describes the way of incorporating a real-time interactive visualization into intelligent agent.

Keywords Intelligent Agent, Information Visualization, Interactive Visualization, Runtime Visualization

1. Introduction

Along with the invention of graphics workstations in mid 1980s, graphics software applications such as computer-aided design (CAD) software became available in the market. Those earliest applications employed graphics technology to provide users with a Visual Human-Computer interface (HCI) for the engineering designs in some specific areas. After graphic-based operating systems were developed for personal computers such as Mackintoshes and PCs in earlier 1990s, visualization became very popular in the design of computer programming environment. For example, Microsoft visual programming environment provides powerful functions to efficiently assist users creating HCIs. In the middle 1990s, Internet has become the most popular information resource that is available for access everywhere in the world. Web browser provides client-side browsing facilities for users to retrieve information through the Internet and display the information in numerous types (such as text, picture, image, audio, video, ...). Information visualization becomes a hot research area and a large number of tools [HEC98] [CK97] are developed to help web users to view, understand and navigate information space. These techniques include scientific visualization [EW92], data mining visualization [FKZ97], visualized concept map [GS95], and many others.

Research in intelligent system can be traced back to the earlier 1970s. The oldest intelligent systems called the expert system developed to address problems in some special problem domains [SE76][MJ83]. Since that, a large number of intelligent principles, methods, and tools covering computer reasoning, computer decision-making, computer learning, and knowledge base are proposed. They were soon evolved to a very large research area --- Artificial

Intelligence (AI). Because of the availability of computer networks and Internet in middle 1990s, research in distributed intelligent system [CMMM92] became a major area in AI. An autonomous and cooperative intelligent system --- intelligent agent [MP93]--- that worked as a human agent to solve problems in a specific environment is the most important research topic in the area of distributed AI.

An *intelligent agent* (IA) or *agent* defined in this paper is a goal-driven and problem-solving computer component that is capable of autonomous, cooperative, and flexible actions to achieve goals. An *action* is a function or method that is written in computer programming language such as Java. Like a human agent, once a goal is given, an agent has its own ways --- plans --- to achieve the goal. A *plan* in an agent is a logical sequence of actions. Unlike a human agent, a software agent provides the procedure and middle results to the user after the goal is achieved.

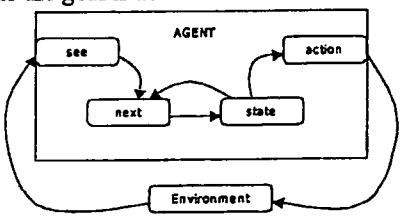

Figure 1: Agents that maintain state [adopted from GW99]

The agent can automatically achieve the goal via its inner state transitions. Functions such as reasoning, decision-making, and interaction are built to support the state transitions in an agent. Figure 1 [GW99] illustrates how agents that maintain state. Because agent users are likely to know the middle stages of agent works, agent systems in nowadays try to provide middle results to users in many ways.

However, most of existing IA systems as described above are not good enough and inadequate for users to operate. This is because:

- The user may like to know the real-time process of the goal achievement. For examples, the user likes to know what is the current process stage, which plan is chosen to achieve the goal currently, what other agents are the partners of this agent and which one is currently communicated, and so on.
- The user may want to be involved in the middle of process. For an instance, if a user believes the current plan that the agent chose is not a good plan to achieve a goal, he/she may want to interrupt the process immediately to help the agent to choose a good plan.

Visualization is the most advanced method because it can show the runtime works of an agent vividly by using graphics and animation. AgentBuilder [RS99] provides visualized output interfaces to display the runtime interactions from one agent to another that has excited the agent users because they are aware of which agent is the agent interacting with at real-time. However, such visualized interfaces have not been used to express the internal state transferring of intelligent agents.

Interactive real-time visualization (IRTV) is a branch of technology that can be used in a computer component to provide graphic-based human-computer interface. By using IRTV as part of the IA system, the running situation of the component can be visualized on-line on a standard output device such as a computer monitor allowing users to understand what the component is doing at any time. Meanwhile, the user may interrupt the running of the component to redress the running of the component if the user believes that the component went in a wrong way.

There are two reasons that motivate us to incorporate IRTV into agents. Firstly, users want to understand the runtime works that are performed by an agent so that they can evaluate the performance of the IA, and IRTV can provide such functions. Secondly, users may like to interact with an IA at be runtime to rectify the running of the agent if they feel the process efficiency is low or the agent could fail to achieve a goal. Therefore, our goal is to provide a general runtime interactive visualization for generating visual output and on-line control and interrupts of the IA.

This paper describes a way to incorporate IRTV in IA. Firstly an agent architecture for hybrid BDI (Belief, Desire, and Intention) is introduced. Then, an approach that is used to visualize the runtime state transitions --- visualize the output --- of an agent is described. Thirdly, an approach that allows an agent accepts users' interrupts and recalls its processes to suitable stages --- visualize the input --- are discussed. Finally, we introduce our future work.

2. An Hybrid Agent

Our agent as illustrated below uses a conceptual model adapted from BDI (Belief, Desire, and Intention) model [RG92]. This model describes reasoning from messages to beliefs (reactive), from beliefs to services (reactive), and from beliefs to goal (deliberative). If a plan goal matches the process goal, then the plan is chosen to achieve the process goal. A commitment is generated to schedule the actions of the plan. All actions are executed according to the commitment.

Figure 2 shows that agent automation starts from coming messages that could be text command from a user, a button being clicked by a user, or a message sent by other agents. An agent is controlled by a top-level algorithm *hybrid_BDI_control* (Figure 3) [AL00]. This main function includes perceiving, reasoning, decision-making, and even interacting sub functions. It first detects the coming messages in a certain period of time. Those messages are put to the correspondent message queue according to triggers. After messages come to message queue, they trigger reactive reasoning, and then deliberative reasoning. After the goal has been achieved, the function drops all successful and impossible messages, beliefs, goals, and plans. Unless an agent is terminated manually, this control algorithm keeps running.

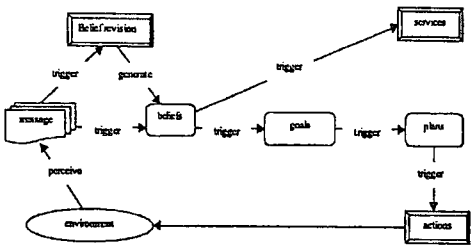

Figure 2: The conceptual architecture of an agent

```
hybrid_BDI_control
{
        initialize_state();
        while (true)
        {
                //messages from user or button
                Messages ms = readMessage(system_message_queue);
                //messages from agent system
                Messages ms = ms ∪ readMessage(agent_system);
                //messages from normal email system
                Messages ms = ms ∪ readMessage(email_system);
                for_each(m ∈ ms)
                {
                        Triggers ts = getTriggers(m);
                        for_each(t ∈ ts)
                        {
                                m_q = toMessageQueue(m, t);
                        }
                }
                for_each(m ∈ m_q);
                {
                        ts = getTriggers(m);
                        for_each (t ∈ ts)
                        {
                                if(beliefRevision(t) != null)
                                        bs = beliefRevision(m);
                                else if (workspace(t) != null)
                                        messageToWorkspace(id, m);
                                for_each(b ∈ bs)
                                {
                                        services = reactiveReasoning(b);
                                        for_each(s ∈ services)
                                                activate(s, message);
                                        actionSeq = deliberative_reasoning(b);
                                        commitments = ActionSchedule(actionSeq);
                                        for_each(action ∈ commitments)
                                                execute(action);
                                }
                        }
                }
                dropSuccessful(M, BEL, G, PL); //BEL: beliefs, G: goals, PL: plans
                dropImpossible(M, BEL, G, PL);
                sleep(2 minutes);
        }
}
```

Figure 3: The top-level control algorithm of an agent

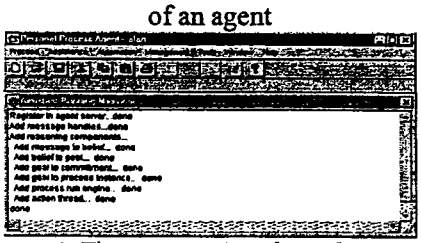

Figure 4: The common interface of an agent

Our agent is written in Java and based on the conceptual architecture and control algorithm described above. Normally, when achieving a goal, more than one agent was constructed. They may situate in the same environment and be responsible for a specific set of events in the environment. Different agent could be generated with different plans and actions (that can be built in design time as well as running time), so different individual agent could have

different abilities (the combination of plans and actions). The common interface of our agent is shown in Figure 4.

3. Output Visualization

We use graphics and animation to present the agent inner state transitions. Firstly, a set of display visual elements --- graphic entities --- that represent different concepts such as agents, messages, believes, goals, plans and actions are defined. Then the graphic links that present the relationships between those graphics are defined. Eventually, this graphic diagram --- visualization --- is used to visually present an agent running situation.

3.1 Visual Concept Elements

The concepts and actions in our agent system are represented with visual pictures. Figure 5 lists most of the pictures of agent concepts:

- AE: An *agent environment* (*environment*) is an environment that an agent situates in and the agent is looking after. An agent environment could include message communication channels, other agents in a multi-agent system, specific databases, and so on.
- Agent: An *agent* is a computer component that situates in an agent environment and that is capable of autonomous and flexible actions to achieve goals.
- Message: A *message* is a piece of information passed from one agent to another. A message in our agent system is wrapped by using ACL protocol [FIPA98].
- Belief: A *belief* in our agent is a statement that the agent believes it is true.
- Goal: A *goal* is an outcome that an agent is going to achieve. A goal could be achieved or failed.
- Plan: A *plan* is a logical sequence of actions.
- Action Queue: An *action queue* (*actions*) is a queue used to store actions.
- Schedule: A *schedule* is a timetable in which each action is associated with a predefined time.
- Action: An *action* is a function or method that is built in computer programming language and can be executed by computer system.

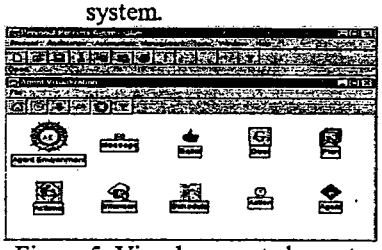

Figure 5: Visual concept elements

3.2 Visual Relationship Elements

Relationships exist between agent concepts. For example, a "perceiving" relationship connects concepts from "Agent Environment" to "Message", i.e., message comes from perceiving agent environment. Similarly, a "reasoning" relationship connects the concepts from "Goal" to "Plan". A relationship is an action that converts the input concept to output concept. If a relationship is represented with an edge, it is a directed edge. For example, the "reasoning" relationship could not connect the concepts from "Plan" to "Goal". The relationships in our agent system are also represented with visual pictures. Figure 6 shows most of them:

- Perceiving: *perceiving* is a relationship that connects the concepts from "Agent Environment" to "Message"
- Belief revision: *beliefrevsing* is a relationship that converts the concepts from "Message" to "Belief"

- Trigger: *trigger* is a relationship that converts the concepts from "Belief" to "Goal"
- Reasoning: *reasoning* is a relationship that converts concepts from "Goal" to "Plan" and from "Plan" to "Actions"
- Decision: *decision* is a relationship that converts concepts from "Actions" to Intention
- Scheduling: *scheduling* is a relationship that converts concepts from "Intention" to "Schedule"
- Pick up: pickup relationship converts concepts from "Schedule" to "Action"
- Doing: *doing* relationship converts concepts from "Action" to "Agent Environment"

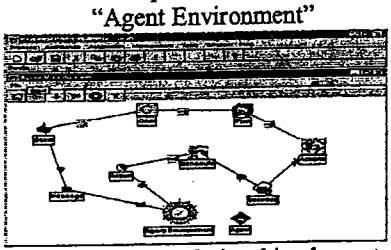

Figure 6: Visual relationship elements

3.3 Structures and APIs

Two types of visual entities, *Icons* and *Links*, are used in the visualization system to present the relational structures of inner state transitions, where Icons present the *states* and Links present the *transfers*. In the visualization, each agent concept is defined as a "state" and each relationship is defined as a "transfer". Those definitions are compatible with the principle of agent achieving goal via its inner state transfers. A *state* has attributes such as *name, geometric coordinators,* and *graphic attributes* associated with its graphic entity "Icon". A *transfer* relationship has attributes such as the *start point, end point, direction,* and *graphic attributes* associated with its graphic entity "Link".

The basic APIs (Application Programming Interface) provided by the general visualization system are:

- AgentVisulization()
  Any agent can generate a visual frame by creating a AgentVisualization object
- addState(String name, int x, int y, String imagename)
  Add a state in an agent system
- addTransfer(String from_name, String to_name, String imagename)
  Add a tranfer in an agent system
- fireTransfer(String from_name, String to_name, String imagename)
  Display an animated connection from "from" state to "to" state with the specific image

3.4 Agent Hybrid Control Algorithm with Visualization

For incorporating visualization into our agent, we changed the hybrid_BDI_control algorithm (Figure 7). Firstly, an agent visualization frame is constructed; then the states that are associated with visual icons are added to a state queue; and finally we draw a link when a state transfer in an agent is taken place.

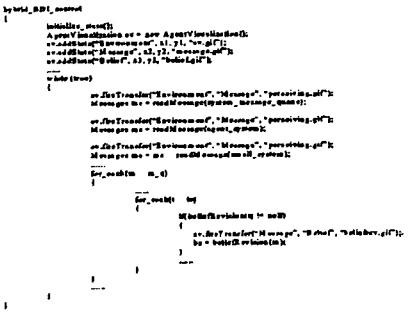

Figure 7: The BDI control algorithm for visualization

4. Input Visualization

There are two major components in the interactive visualization system. The first component as described in last section is used to visualize the agent output. The second component is used for on-line manipulating of the workflow. The system accepts users' real-time interrupt --- input and then backward the current process of

agent to a suitable stage. A stage must consist of at least one transaction (Figure 8). A *transaction* in an agent is a set of actions that cannot be executed separately. For example, a belief revision is a transaction including actions such as picking up a message, parsing the semantics of the message, changing associated beliefs, and then save those beliefs to belief library. An agent deals with a user's interrupt after a transaction is completed. If an interrupt event happens in the middle of a transaction, it is accepted and saved in an interrupt event queue. All transactions related to a goal are saved in a persistent transaction queue. After the goal is achieved, those transactions are moved to a transaction library.

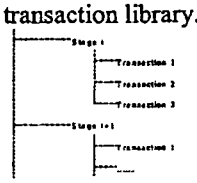

Figure 8: stage and transaction

A "Pause" button is provided in the toolbar of the agent visualization frame (Figure 9a). When click this button, if the agent is working in a transaction, it will keep working until the transaction is completed. Otherwise, it will stop its work immediately. A "Back" button is provided in the same toolbar (Figure 9b). After an agent stops its work, the "back" button is activated. One click of "back" button will cause one transaction backward. All states and transfers of agent are going back to that stage when the process is backward to a stage.

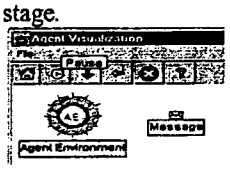

Figure 9a: Pause button

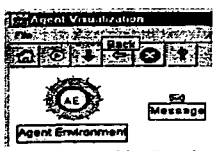

Figure 9b: Back button

Nothing could be more understandable by showing an example to explain the input visualization functions in this stage. Suppose that a user (a Ph.D student) uses an agent to help him finding fund (a goal) to support attending an oversea conference. The agent has three plans to achieve the goal. Firstly, the agent could send an email to ask the student's supervisor; secondly, it can send email to ask the student's department; and finally, it can send email to ask the faculty. For some reason, the agent believes "ask department" is a good plan on its own way (Figure 10). The user, however, thinks this plan is not good because he/she knows that his/her supervisor is going to fund him/her. Then the user could interrupt the agent's work and ask the agent back to goal state, and then choose a plan for the agent (double click the "ask supervisor" plan icon). After the "good" plan is selected by double clicking its icon in the agent visualization frame, and then click "Go" button, the agent continues its work using an "input" plan (Figure 11).

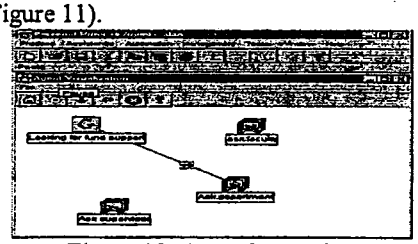

Figure 10: Agent has a plan

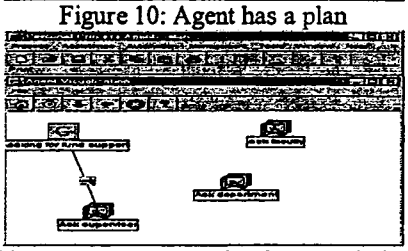

Figure 11: Agent running in a good plan after user's input

5. Future Work

IRTV provides a visual approach to direct manipulate the agent's workflow and observe the agent inner state transition (its runtime work) with a icon-link based visualization. By using this graphic user interface, a user can correct the agent working flow by on-line visual interactions. Agent users are able to improve the agent performance to achieve the goal by the assistance of visual interactions. The design and implementation of those methods are simple because it looks like to replace the

traditional text input and output with the graphic input and output.

Our future research focuses will be on agent visual programming, i.e., we try to integrate visual input and output with the computer programming language such as Java. The future research will result in a new Java package. The output and input APIs in that package will be basically associated with visual interfaces instead of text-based interfaces. We will also investigate the layout algorithms which will provide better layouts for these icon-link based visualizations that address the problem of human comprehension of diagrams.

References

[AL00] Aizhong Lin. Multi-agent Business Process Management. Proceedings of ISA'2000, International ICSC Congress on INTELLIGENT SYSTEMS AND APPLICATIONS on December 11-15, 2000, University of Wollongong , NSW, Australia

[CK97] Jeromy Carriere and Rick Kazman. WebQuery: Searching and visualizing the Web through connectivity. In Proceedings of the Sixth International World Wide Web Conference, 1997.

[CMMM92] B. Chaib-Draa, B. Moulin, R. Mandiau, P. Millot. Trends in distributed artificial intelligence. Artificial Intelligence Review 1(6), 6(1), pp. 35-66, 1992.

[EW92] Earnshaw, R.A., Wiseman, N. An Introductory Guide to Scientific Visualization, Springer-Verlag: Berlin, 1992.

[FIPA98] FIPA specification. Agent Communication Language. http://www.fipa.org/specs/fipa00003/OC00003A.html

[FKZ97] Ronen Feldman, Will Klosgen, and Amir Zilberstein. 1997. Visualization techniques to explore data mining results for document collections. In Proceedings of the Third Annual Conference on Knowledge Discovery and Data Mining (KDD), Newport Beach.

[GS95] Brian R Gaines and Mildred L G Shaw. Collaboration through Concept Maps. http://ksi.cpsc.ucalgary.ca/articles/CSCL95CM/

[GW99] Gerhard Weiss (Ed.). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (1999) pp27-77

[HEC98] Mao Lin Huang, Peter Eades, and Robert Cohen. WebOFDAV - navigating and visualizing the web on-line with animated context swapping. Computer Networks and ISDN Systems. Special Issue on the 7th International World Wide Web Conference, 30(1-7):638-642, 1998.

[MJ83] M. Mauldin, G. Jacobson, A. Appel and L. Hamey. ROG-O-MATIC: A Belligerent Expert System. Technical Report CMU-CS-83-144, Carnegie-Mellon University, July, 1983.

[MP93] Jörg P. Müller and Markus Pischel. The Agent Architecture InteRRaP: Concept and Application. Research Report, Deutsches Forschungszentrum für Künstliche Intelligenz, Number RR-93-26, p. 99, 1993.

[RG92] Anand S. Rao and Michael P. Georgeff, An Abstract Architecture for Rational Agents, in Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, edited by C. Rich, W. Swartout and B. Nebel, Morgan Kaufmann Publishers, San Mateo, CA, 1992

[RS99] Reticular System Inc. AgentBuilder® White Paper. http://www.reticular.com/Library/white_paper_r1.3.pdf

[SE76] Shortliffe, Edward H. (1976). Computer-Based Medical Consultations: MYCIN, American Elsevier, New York, NY