An Intelligent Believable Agent Environment

Guy Taylor and Lin Padgham
Department of Computer Science
RMIT University, Melbourne, Australia
email: linpa@cs.rmit.edu.au, guy@yallara.cs.rmit.edu.au

Abstract

In this paper we present an intelligent believable agent development environment. This was achieved by connecting dMARS, an agent-oriented system, and Open Inventor, an animation system. We describe the design of this environment which involves a connecting layer and some plan templates for (i) managing the synchronisation between actions determined in the reasoning system, and their execution in the virtual world of the animation; and (ii) managing the physical aspects of the agents, including sensing. Emotions and personality are also modelled to some extent. We successfully implemented a real-time animation using this environment which we call Dog World, with dogs which behaved in interesting ways, reasoning and interacting with their environment. The merits of the system are evaluated conceptually and a comparison is made with two other believable agent systems.

1 Introduction

This paper describes an environment which we have developed for modeling *intelligent believable agents* in a dynamic world. Intelligent agents (in a computer science context) are robust software modules which drive conceptual entities, causing them to behave rationally, in a complex and dynamic world. Believable intelligent agents require these conceptual entities to in addition be engaging and believable. A necessary criterion is that the agents and their behaviours can be directly perceived in real time. We achieve this by using a graphical animation system to represent the agents visually. We combine this animation system with a rational agent reasoning system, giving a platform where we can experiment with various aspects of believable agents, such as personality and emotions, and interaction with the environment.

2 dMars and Open Inventor^{TM}

The two systems used for creating our intelligent believable agent environment were the rational agent system, dMars[1] (distributed Multi Agent Reasoning System), an updated version of PRS (Procedural Reasoning System) [GI89, IGR92], and the object oriented animation system Open Inventor^{TM}[2]. We give a brief overeview of these two systems in order to set the stage for the description of our environment.

2.1 dMars

PRS, the precursor to dMars, and dMars itself, have been used in a number of demanding real-time applications, including an application automating the monitoring of the Reaction Control System of NASA's space shuttle, an airport traffic control system, and a system for supporting service staff of a large telecommunications company.

The system is based on a cognitive model of beliefs, desires and intentions which determine what any agent does at a particular time. Each dMARS agent consists of a database of current beliefs or facts about the world, a set of current goals to be realized, a set of plans describing how a sequence of actions may achieve a given goal or be a reaction to a particular situation and an intention structure containing a set of plans that have been chosen for eventual execution. An inference mechanism manipulates these components, selecting appropriate plans based on the agent's beliefs and goals, placing those selected plans in the intention structure and executing them. The system interacts with its environment through its database by acquiring new beliefs and through the actions it performs.

Plans are complex specifications of how agents can achieve goals. They may be hierarchical in nature, creating subgoals which must be realised as the plan is carried out. Plans consist of a body, which describes the different steps in the procedure, and an invocation consisting of a triggering part and a context condition part. The triggering part describes the events that must occur for the plan to be executed. They can be the acquisition of a new goal or some change in the belief database. The context condition describes conditions which must be satisfied in the belief database. A dMARS agent decides which plans are applicable by matching beliefs and goals with invocation conditions. For more information about how the inference mechanism selects the applicable plans, manages them in the intention structure and decides on execution we refer to [GI89, IGR92].

[1] From the Australian Artificial Intelligence Institute, AAII

[2] Trademark of Silicon Graphics

2.2 Open Inventor^{TM}

Open Inventor^{TM} [Wer94, SC92] is an object-oriented animation system providing a toolkit which allows the creation of interactive 3D graphics. It is composed of libraries of graphical primitives each with their own attributes and methods.

Most graphical applications have to deal with the drawing of graphics and the methods which achieve this. Open Inventor^{TM} supports this by encapsulating all the methods for objects, such as displaying, manipulating etc., into the objects themselves. User defined methods can also be added. For example, to create a cube, one simply creates an instance of a cube, defining it's attributes and Open Inventor^{TM} will then take care of drawing it. If the view is changed Inventor will re-calculate the cube's appearance and re-display it.

Inventor creates a scene graph of all the objects in the environment, which contains the objects and any transforms that might be applied to them. Information can be added to this scene graph either by C++ code which creates or manipulates objects, or by reading in a model file.

The ability to load model files is particularly useful as it allows a graphical model of an agent to be created using a 3D drawing tool. The agent can be built up out of simple graphical primitives, such as cubes, cones and spheres, thus allowing any visual changes to the agent to be handled by Open Inventor^{TM}'s built in methods for these primitives.

Open Inventor^{TM} also allows parts to be dynamically connected to each other. For example a dog tail can be connected to a dog body via a dynamic connection (known as an engine) which allows the tail to move in relation to the body. These dynamic connections can also be specified in the model file, and turned on and off when desired. For example, a motion path for a dog tail can be specified in the model file as the engine between the tail and body, allowing a wag action by turning on this engine.

The system also provides facilities such as ray intersection objects which are useful for modeling accessibility in an area relative to an object. For example sight can be modeled using a ray intersection object emanating from the agent's eye. (This will return a list of objects that are visible along a 3D vector with starting point at the agent's eye).

The provision of high level methods and modeling constructs greatly simplifies the specification of the animation aspects of the agents and their environment.

3 System Design

The successful integration of dMars and Open Inventor^{TM} has involved using the 'Foreign Process Agent' supplied by dMars, which allows communication with dMars as if it was another agent, but allows application code to be connected to the Foreign Process Agent under control of the application. The Foreign Process Agent also supports communication via a socket, allowing us to use separate hardware for the animation and dMars software.

We have implemented a *connecting layer* which uses the Open Inventor^{TM} library and is connected to dMars using the Foreign Process Agent. We also found it necessary to design a set of plan templates to be followed in writing dMars plans that involve actions where the time taken to execute the action in the animation world must be taken into account.

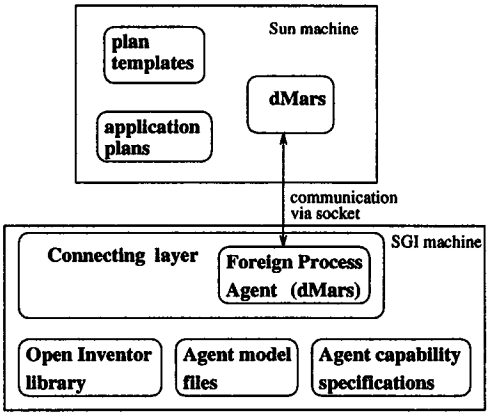

Figure 1: Architectural overview.

We give an overview of the architectural structure in figure 1, and then describe in more detail the connecting layer and the plan templates.

3.1 Connecting Layer

Open Inventor^{TM} models both the physical environment and the physical bodies of the agents, while dMars models the reasoning processes for each agent. The connecting layer includes two major subsystems, one for managing actions - how the bodies of the agents move within the physical environment, and are synchronised with the reasoning processes in dMars; and one for managing senses - how the agents percieve the environment, and communicate these perceptions to the dMars reasoning processes.

3.1.1 The action subsystem

The action subsystem of the connecting layer enables high level actions, appropriate to the reasoning system to be transformed to the primitive actions needed for the animation system.

There are three different types of actions that were identified and catered for. The simplest are atomic actions which are conceptually instantaneous and have a negligible duration. As far as the reasoning system is concerned they can either succeed or fail. These actions are at the same level of granularity as atomic actions in the animation system and can be mapped to a single command in the animation system. An example is a turn action, which would be mapped to a rotation.

The other two kinds of actions have duration. They can either have a known end condition such as an action to move to a particular location, or they can have no fixed end point such as the action to wag a tail.

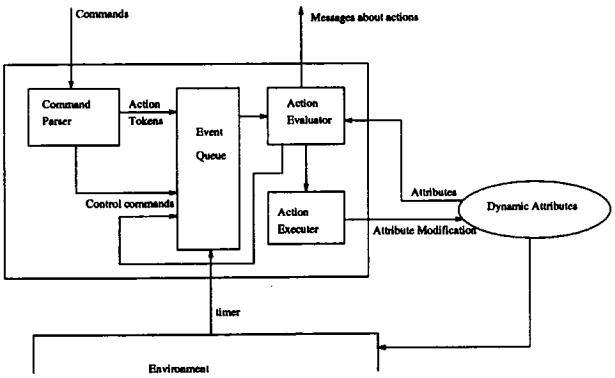

Figure 2: Action section of Joining Layer

A figure of the architecture for the action subsection of the connecting layer is shown in figure 2.

The Command parser looks at commands received from the agent layer, of which there are two types, control commands and action commands. It then translates them into 'action tokens', which are the actions matching the granularity of the animation system.

Action commands are mapped into one or more action tokens which are placed in the queue. An action token is the lowest level graphical action and is atomic in nature, taking a single time frame to realize.

Control commands are used for controlling the queue in ways other than the normal execution. In order to allow the system to be fully reactive, new goals/intentions must be able to take priority over current goals at any point in time. This requires that executing actions must be able to be aborted and/or suspended, at the animation level as well as the dMars level. Currently, only flushing is supported, allowing the reasoning system (agent mind) to tell the physical system (agent body) to abort the current action.

The Action evaluator checks the validity of the next command token. The reasoning system (agent mind) might start an action, but some time during its execution a condition could arise which means the action is no longer able to finish executing. In this case the action evaluator sends a message to the agent and the current activity is aborted. Thus the action evaluator checks the agents capabilty with respect to the action, at each step.

Finally, the action token executer sets the values of the graphical models in the world at each timer event by reading the head of the event queue. Each action token has associated with it a motion path, allowing Open Inventor^{TM} to create a smooth animation for that atomic action. For example an action like 'take a step' can have the motion path for leg movement included in the model file, allowing smooth walking to be easily realised.

3.1.2 The sensing subsystem

The other subsystem of the connecting layer is the sensing subsystem. Senses are not modeled at all in dMARS but are an important part of fully believable agents. It is appropriate to model senses in the connecting layer as they are part of the agents physical body but also have a direct influence on the agents reasoning.

Two types of sensing are modeled - goal-directed sensing and automatic sensing. Goal directed sensing allows an agent to explicitly ask for a piece of sensory information, receiving a response from the connecting layer. An automatic sense is triggered whenever a certain condition occurs. For example smell may be modeled as an automatic sense. As soon as an object is within range of the agent's smelling capability, a message would be triggered, alerting the agent to the newly found smell.

An automatic sense is set up by indicating what conditions would trigger the sense, for the given agent. For example smell may be triggered by any object with a smell attribute, within a certain radius of the agent. The connecting layer then monitors for the occurrence of this situation.

At each timer event, the connecting layer checks to see if something is sensed, based on the agents capabilities and the current environment. If the sensory information is different from that of the last timer event, then a message is generated and sent to the agent. This mechanism avoids redundant messages when sensory information remains unchanged.

3.2 Plan Templates

As mentioned previously, three distinct types of actions were identified, with respect to this environment - atomic actions, durational actions with indeterminate end-point, and durational actions with fixed end-point. Atomic actions are non-problematic because of the match in granularity between the two systems. Durational actions with indeterminate end-point can be fairly trivially mapped to pairs of atomic actions that are to start and stop the action. Durational actions with fixed end-point are however more complex to manage. We have developed a template to be used for the dMars specification of plans involving these actions. Use of this template simplifies the appropriate co-ordination between dMars and Open Inventor^{TM} in the management of these actions.

The issue is that in order to keep the two systems synchronised, it is important for dMars to know when a durational action has been finished, allowing further processing of subsequent actions. The processing of a durational action such as 'move to a location' takes negligible time at the dMars level, compared to the time it takes to execute in the virtual world of the animation system. Thus a mechanism must exist for the dMars processing to suspend until notified by Open Inventor^{TM} that the physical action has completed.

To facilitate the necessary synchronisation process a set of plan templates have been developed for specifying standard durational actions. There are three plan templates - an action initiation template, a successful completion template, and a failed completion template. For each standard durational action in a dMars plan, three sub-plans following these templates must be created.

The action initiation template plan has as its invocation condition, the posting of a goal which corresponds to the action. For example, the move agent to an object action has an initiation plan which is invoked by the posting a `(move-to $obj)` goal.

The action initiation plan starts by asserting a belief which is a description of the particular action, (for example `state moving`). The next step in the initiation plan is to tell the connecting layer (through a message) to do the action. Following this is a wait action, which suspends the plan until the current state belief is negated.

When a sequence of action tokens constituting a dMars action is finished in the animation level, or if an action token fails due to the action evaluator, then a message is sent to dMARS informing it of the event. This message triggers either the plan for success or the plan for failure. Both the success and the failure plans negate the state belief, thus causing the suspended wait action to finish and the initiation plan to terminate.

This set of templates provides a mechanism for implementing standard durational actions. A goal is posted to perform the action and will not complete until the action at the animation layer finishes. If another action command is posted whilst an action is executing (due to a higher priority action being required) then the goal plan will be aborted in favor of the new action.

4 Dog World Prototype

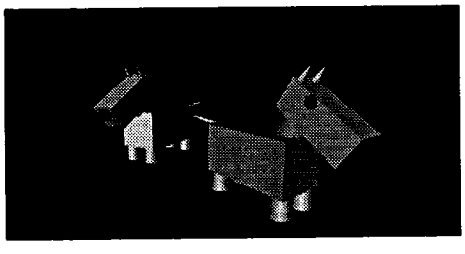

Figure 3: Two dogs in Dog World

We have developed a sample scenario which we call Dog World as a prototype application using this environment. This world is relatively simple. It is a large square region that contains both dogs and bowls of dog food. The dogs are agents while the food is part of the environment. Figure 3 shows a screen dump of 2 dogs modeled in this world. One of the dogs' desires is to find and eat food.

The dogs have a sense of smell with which they can locate objects, namely food and other dogs. Their sense of smell is limited to a certain radius around the dog.

Dogs also have emotions of aggression, fear and hunger which can affect their behavior. The dogs become more hungry as time goes by, and less hungry as they eat. A dog is capable of performing actions to walk, eat, bark, wag his tail and run away from a threat.

How a dog behaves at a certain time depends on a combination of it's goals, it's internal belief states and it's emotional states of aggression, fear and hunger. Emotions are modeled in the reasoning system by varying the quantities of aggression, fright and hunger.

Aggression is the dog's current aggression level and aggression_threshold is the point at which the dog's behavior can be considered 'aggressive', the same applies for fright. The combination of these represents the dog's emotional state. A dog with a high aggression level and a low fear level would behave differently from one with the opposite set of values.

The food and food_threshold correspond to the dogs hunger level, i.e. a small food value means that a dog is hungry. The food_threshold is used as a level at which the dog starts to become aggressive due to not eating.

Different personalities can be modeled by varying the threshold where these quantities start to take effect. In this way, dogs modeled using the same set of plans will act differently just by changing these thresholds. Rate of change of hunger/aggression/fear can also be modified to give agents differing personalities.

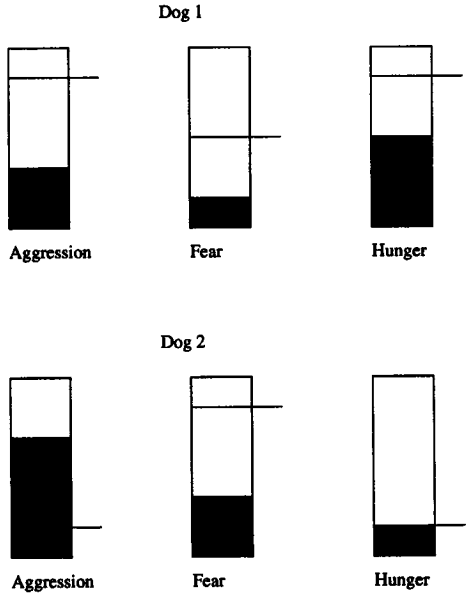

Figure 4: Personality using emotions.

To illustrate this point, in figure 4, there is an example of two dogs. The solid portion represents the current value for that quantity whilst the line coming out of the right hand side represents the threshold which needs to be reached to trigger the reaction of that emotion. Dog 1 can be viewed as a passive dog

who takes a lot to anger but is not too hard to scare. It will also only become aggressive when very hungry. Dog 2, on the other hand, is very aggressive and is very hard to scare, even if remotely hungry it will start to become aggressive quickly.

A final point is the concept of time. The connecting layer manages a timer based on one click per frame, so that things that change over time can be modeled. In the Dogworld scenario the hunger level rises over time.

A dog can be motivated in two ways, either through goal-directed events or by automatic sense events. The goal-directed behaviors are what the dogs decide to do themselves. In the plans, if a dog is idle, then it will invoke the find-food goal. This does two things, if there is food in smell range, the dog will move to it and eat it. If there is none, then the dog will wander around looking for some.

The rest of the dog's behavior is modeled through automatic sense events. There are three possible events, food or dogs coming into range and being barked at. The simplest is when food is sensed. The behavior is then exactly the same as the goal directed event, the dog will move to the food and eat it.

When a new dog is sensed different behaviors will result depending on the dog's aggression level. If it is passive then it will start to wag its tail, if it is aggressive then it will start to bark at the other dog. Being barked at has numerous behavioral reactions depending on the emotional state of the dog. If the dog is aggressive, it will bark back. This can result in a "barking match" between two dogs. If a dog is frightened, then it will run away from the barker. If it is neither, then it will simply ignore the other dog. When a dog is barked at its fright level goes up, and when it barks, its aggression goes up. In this way if a "barking match" is started the first dog to become frightened will end up running away.

Even this very simple Dog World allows the creation of rich and varied animations with considerable complexity. By combining all the above elements believable dogs were created which appeared to have different personalities.

5 System Comparison

The environment we have created does allow for the development of intelligent believable agents - intelligent because they exhibit complex rational behaviour, believable because they exhibit differing "personalities" and because their physical behaviour can be seen in a real-time animation. We compare the system we have created to two of the most well known believable agent systems in the literature, the SodaJack Project (part of the AnimNL project) [LB94a, LB94b], and the TOK system [Bat94, BLR92b, RB92, KWB92, Bat93, LB91, BLS92, LB93, BLR92a].

These three systems all share significant elements in common. They all have a goal-based reasoning system which drives an animation. The reasoning system in TOK and the ItPlanS system used by Jack are similar but less sophisticated versions of agent oriented reasoning systems. For example the Jack system does not have advanced modelling features such as maintenance conditions, and TOK does not allow the same complexity of plans as does dMars.

One useful feature of TOK is the ability for an agent to recognise when its current goal has been achieved due to a change in the environment (which may have been caused by another agent). In our system this recognition can only be achieved by carefully hand crafting plans and their interactions.

One of the issues addressed to some extent by all three systems is that of achieving smooth animation. However the systems approach this problem in quite different ways. The TOK system anticipates the next move prior to the current animation sequence's completion. In this way the motor section can be aware of what it is expected to do next and it allows for a smooth transition as motion can be continuous from one action type to another. Smooth transitions are a natural by product of the method used by the Jack system. Inverse-kinematics and dynamics are used to simulate the way that Jack moves.

In our system a simpler approach was taken, using a 'neutral pose'. Any action which took the agent away from it's natural pose would then place it back in it on completion, thus avoiding jerky transitions. This is obviously not as sophisticated as the above methods but has proved to be adequate so far.

Another important constraint on actions, is their feasibility. For actions to seem realistic they must be able to fail in certain situations. This can happen in two ways, one is when multiple actions are attempted which can't be performed at the same time, the other is when some environmental condition disallows an action.

The TOK system avoids situations like walk left, together with walk right by using a notion of body resources, which can only be used by one action at a time. TOK also has explicitly disallowed combinations of actions. The Jack system has a section of the OSR devoted to feasibility checking. If there is a failure this is reported to the planner to allow it to formulate a new plan.

Our system takes a different approach for disallowing use of the same resource by using the maintenance field. For example, if a dog is performing an action, then a state will be asserted which represents what is being done. If another action is desired then the state is checked. If this is incompatible and if the agent still asks for the action to be performed, it will then abort the previous action as there is a conflict with the maintenance condition.

Sensing is handled differently in the three systems. Jack has only a limited form of sense capability which is modelled as requests to the environment. The TOK system uses a more advanced method of sensing, where

sensors are attached to plans and are turned on and off as the plans are invoked. These sensors are very low-level and report information such as 'can I see woggle[3] X jumping?' [LB93, page 4].

In our system conceptually higher level senses are explicitly modelled in the connecting layer and attempt to approximate real senses such as smell. These are always active and report messages back to the agent when relevant new conditions arise. These conditions are specified by an agents sensory capabilities which are a part of the declarative information about an agent.

The final comparison involves the way emotions and personality are handled by the systems. Jack does not really model emotions, as it is intended more for simulation of characters. The TOK system has complex emotional process. These are modelled by passing success and failure results to Em (Tok's emotional subsection). This takes these results and infers emotions from them, e.g. a failed plan causes unhappiness. Our approach is simpler, as we let emotional quantities vary according to different actions. This is not as powerful as a full emotional simulation, however complex emotional behaviour perceived as different personalities can occur.

6 Discussion and Conclusion

The system described here provides a base for creating and manipulating powerful intelligent and believable agents, in a relatively simple way. This environment provides a tool for further research exploring such things as aspects of personality, interaction of deliberative and reactive behaviour and interaction of personality types in a multi agent system.

Believable agents is a relatively new research field, and as a result there are not yet many implemented systems/environments available. Comparison of our system with the two main systems discussed in the literature indicates that our system provides more complex reasoning (using dMars) than the other systems, and a more intuitive modeling of senses. However TOK has a more complex emotional component, and Sodajack has more complex animation.

References

[Bat93] J. Bates. The nature of characters in interactive worlds and the oz project. *Virtual Reality: Anthology of Industry and Culture*, 1993.

[Bat94] J. Bates. The role of emotion in believable agents. *Communications of the ACM (Special Issue on Agents)*, 1994.

[BLR92a] J. Bates, A.B. Loyall, and W. S. Reilly. An architecture for action, emotion and social behavior. *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, 1992.

[BLR92b] J. Bates, B. Loyall, and W. S. Reilly. Broad agents. *Proceddings of the AAAI Spring Symposium on Intergrated Architectures*, 2(4), August 1992.

[BLS92] J. Bates, A. B. Loyall, and W. S. Scott. Interating reactivity, goals and motions. *Preoceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, 1992.

[GI89] M. Georgeff and F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI '89*, pages 972–978, Detroit, USA, Aug 1989.

[IGR92] M. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, pages 34–44, Dec 1992.

[KWB92] M. T. Kelso, P. Weyhrauch, and J. Bates. Dramatic presence. *PRESENCE, The Journal of Teleoperators and Virtual Reality*, 1992.

[LB91] A. B. Loyall and J. Bates. Hap. a reactive, adaptive architecture for agents. *Technical Report CMU-CS-91-147*, 1991.

[LB93] A. B. Loyall and J. Bates. Real-time control of animated broad agents. *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, June 1993.

[LB94a] L. Levison and N. Badler. How animated agents perform tasks: Connecting planning and maniplulation through object-specific reasoning. *AAAI Spring Symposium*, 1994.

[LB94b] L. Levison and N. Balder. Sodajack: an architecture for agents that search for and manipulate objects. *Tech. Rep. MS-CIS-94-16/LINC LAB 265*, 1994.

[RB92] W. S. Reilly and J. Bates. Building emotional agents. *Technical Report CMU-CS-92-143*, 1992.

[SC92] P. S. Strauss and R. Carey. An object-oriented 3d graphics toolkit. In *SIGGRAPH'92*, pages 341–347, Chicago, 1992. ACM.

[Wer94] J. Wernecke. *The Inventor Mentor*. Addison-Wesley Publishing Company, 1994.

[3] Woggles are the creatures in the TOK World.