Chapter 10

Conclusions and Future Work

10.1 Conclusions

In this thesis we have presented a knowledge-based framework for computer animation, focusing specifically on the aspect of behavioural animation. Behavioural animation is concerned with animated entities which can develop autonomous behaviours in dynamic environments. The aim of this approach is to shift most of the control to the entities themselves because dynamic environments involve an overwhelming number of detailed motions which are too difficult to be controlled, or scripted, by human animators. We have thus developed AI techniques to produce control mechanisms that implement autonomous behaviour.

The framework integrates a number of control structures with features stemming from some important requirements:

- use should be made of libraries of motions provided in existing animation systems (Chapter 4) and these motions can be used as basic units in the composition of more elaborate motions;
- the animated entities are to possess autonomy in developing their motion;
- reasoning about actions is a necessary element of intelligent behaviour [Wilk88];
- the role of the animator should be simplified, concentrating on co-ordinating the overall animation and giving minimum directions to individual animated objects.

As presented in Chapters 3 and 6, actions representing behaviours can be regarded as simple statements involving a verb and a number of attributes. The requirements listed above are basically concerned in transforming such statements into full sequences of

known motions that would otherwise be specified by an animator. The implementation of the framework as an animation system composed by two interacting blocks (Chapter 5) can achieve the transformation mentioned above. This separation into two blocks occurs because the block comprising the libraries of motions already exists in animation systems and involves numerical computation which can be adequately programmed in conventional languages (Chapter 5). While the other block known as the *Controller*, which controls the behaviour of the animated entities, deals with concepts or knowledge which are written and processed as symbols in AI languages. The latter block of particular interest in our work and it is the focus of the following discussion.

The *Controller* solves the problem of motion control of the animated entities. The use of the blackboard model to implement the *Controller* permits the structuring of the components of the problem solving into independent but related modules. Both control entities, the *instruction* and the *task,* are types of structures that implement a conceptual model of action which are operated by the respective modules. These modules are procedures of the domain knowledge called *knowledge sources*. The inclusion of *message* as a type of entity further extends the capabilities of the agents to effect interaction between them which results in co-operation. In order to co-ordinate the load of activities of the animated entities, an additional module to schedule future activities is added to the *Controller*. The scheduler starts the instructions in the script in the specified times and holds the excess of actions generated during the execution of the animation.

As described above, each module of the *Controller* implements one specific activity which can be easily identified and modified when needed. The capabilities of the *Controller* can be further extended by adding new modules to the system requiring little change in the overall control. Such a modular organisation permits the components of the conceptual framework be represented as computational constructs and be executed in a specified order.

The concept of the *instruction* is *flexible*, *powerful*, and *uniform* in representing actions performed by animated entities. It is flexible because it can accommodate a number of features found in conventional languages such as recursion and parameterisation, and use them as needed. It is powerful because the representation of an instruction, as being

composed of sequences of instructions and other conceptual structures, permits increasingly complex actions to be represented and to be applied in different contexts. This also emphasises the re-use of existing instructions as a way of saving work. It is uniform because the instructions entered by the animator are similar to those started from within the system during runtime, allowing them to blend naturally. The only distinction between these types of instructions is that they are annotated with the origin of their respective source for the purpose of priority rating.

The animation system serves as a tool that assist the animator in the task of generating animation sequences. As its input the animation system receives the script of high level commands (e.g., 5 *instructions*) prepared by the animator. As its output it generates a detailed script with a sequence of motions commands (e.g., 500 simple motions) that can be readily performed by the animated entities. The animator can modify the script as many time as necessary until a satisfactory animation is achieved. The degree of control over the animation is obtained by modifying the start times, changing the scripted commands and parameters, or specifying more detailed commands in place of a higher level one. At the lowest level, "fine tuning" can be made to the detailed script and then run this directly in the *Basic Animation System*. For example, it is possible to change the definition of the colour of an object or insert commands to set the camera viewpoint and execute the detailed script without generating it again from the animator's script. Fine tuning is an expedient accessible to the animator but it is rarely used. Direct editing of the detailed script is typically used for the post-processing stage of the animation process.

As the animation becomes more elaborate so does the motion vocabulary, the simple adjustment of the animator's script may become ineffective and the re-evaluation of the instruction may become necessary. Some of the animator actions can be: the composition of new instructions or re-structuring the existing instructions at the level of the *Controller*, or the programming of new skills performed by the figures at the level of the BAS. In this thesis we have exemplified a number of instructions reflecting actions employed in a bar scenario which serve as examples for other scenarios. We have also presented a number of features related to its implementations such as recursion, grouping of instructions, parameterisation, etc.

The focus of this work has been on developing a framework that provides the animator with facilities to organise and re-use the high-level motion specification. We employed a Prolog environment for handling symbolic processing, rule inferencing, and interactive capabilities. As an important extension to this work we strongly recommend the building up of a dialogue interface with which the animator could be assisted in the specification and experimentation of new instructions, or its debugging. The problem is not with the underlying language that the animator could not be familiar with, but the number of instructions the animator might have to inspect. Each instruction has a number of parameters, rules, relations, alternative plans, etc. And the hierarchical tree yielded by the execution of high-level instruction is complex. Thus, such an interface could be regarded as an additional layer to be placed on top of the Controller. Consequently the task of the animator is even easier. Instead of recalling the name of a specific instruction and its parameters from his memory, he would simply browse the dialogue interface, select item, and fill slots. As a system developer, the animator may wish to add new instructions and new tasks, or even extend the current instructions with new alternative plans. Further discussion is given in the next section.

This framework has the main characteristics of the Calvert's ideal system [Calv91]. At the highest level, high-level directions are input in the system through a script. At the intermediate level, the high-level input gives rise to a detailed script. This script contains details of movements for the figures to perform which are specified by goal actions: one part originates from the animator's script itself; and the other part originates from the system in the form of default actions, reactions to the environment, and interactions between agents. These goal actions are developed into full plans that generate simple commands to input to the BAS. In order to allow editing these simple commands are stored in a file and run off-line by the BAS. Such a file contains the commands of the lowest level of the ideal system.

10.2 Future Work: The Animator's Interface

Currently the debugging of the animation is very awkward because of the poor interpretative environment provided by the LPA's *flex*/Prolog. The strategy to solve a problem, if an unexpected behaviour occurs, is to insert several break points and obtain

printouts of the procedures that might give clues about the problem. Thus, the animation is run and pauses at the break points when data may be displayed through the use of on-line commands. As the data structures are linked to each other, the links are followed and displayed one by one. Normally this process must be repeated several times before the problem is found!

The experience with the current prototype made us to realise that a dialogue interface would be extremely useful in developing new instructions as well as in their debugging. The envisaged interface should therefore provide facilities that combine both editing of the instructions and monitoring their execution. This is important because a process is composed of pieces of connected instances of data. With such an interface the animator would not need to find each connection by hand. With the click of a button the piece of data would be brought up to the screen without interrupting the program. Thus, the animation program could be run step by step.

The editing capability is important because an instruction is comprised of several pieces of data (rules, relations, plans, and frames) which are scattered in different files, thus an editor would help to bring these loose but related components together, allowing the animator to have an overall view of the instruction under consideration. Thus, the animator would be able to test and organise the instructions, for example, by composing a plan, and modifying its parameters, testing the relations used in context identification, including new slots in the frames, etc. The editor can also be regarded as an interpreter which would call the correspondent KS (instruction, task, or message) to perform a "controlled execution" of an "isolated instruction process". It would allow the links established between an instruction instance and the nodes of its plan to be observed. The possibility of changing values in an instance and retrying it would enable the animator to make a variety of tests. This would be helpful, for example, to spot a wrong parameter in an instruction plan or a mistyped parameter name.

It is equally important that the interface can monitor the execution of the instructions in the actual animation context. Because of the amount of data generated, the interface should organise, in a dialogue window, a number of buttons to display different types of instances: instruction, task, message, root (process), agent, object, etc. A click on a slot of an instance containing a name of an instance would pop up a new dialogue window or

re-use the same one to show the attributes of that instance. For example, starting with a list of agents the animator could select an agent from it and display the attributes of that agent, then the animator can pop up the process currently being executed, and so on. It is also interesting to display the processes which are currently active or idle. Therefore, the interface would be useful for browsing the information about processes and for navigating through linked instances.

Such interactivity would significantly accelerate development work, perhaps by a hundredfold! The fact that animators do not need to guess or to keep in mind the name of the instance would be of considerable help. The browsing of a data attributes together with the display of a 2-D view of an executing instruction is ideal. The interface would help in the planning or in re-structuring instructions. This is particular important as the animation vocabulary evolves. The interface can be the third block of the animation system along with the Controller and the BAS.

10.3 Future Work: Coordination of actions

The ability to organise opportunistically two or more actions into a single coordinated one is an important feature that could be included in the Controller as a new knowledge source or as an additional part of the Scheduler's activity. This feature is intended to provide the capability to the agents to display intelligent behaviour automatically. For example, suppose that a waiter is carrying out his work. He is currently holding one used glass and he is on the way to collect another one, but suddenly he receives an order from a customer to bring a drink. At this point a decision can be made with the purpose of co-ordinating several related actions: either he collects the second used glass because it is quite near and would not cause too much delay before serving the customer; or he could put the glass down on the nearest table or counter, and serve the customer; or he could immediately serve the customer because one of the hands is free. Another example is a barman serving a customer. If another customer arrives in the meantime and places an order, the barman continues to serve the first customer, but instead of bringing one drink he will bring two. Basically it is necessary to identify the activities currently being carried out along with those still to be carried out by the agent. This is very similar to the context identification in the sense that the rule identifies the actions to be performed

and decides on a plan of action. Thus, possible contexts related to an action are grouped into a ruleset.

The identification of possible patterns of actions requires the testing of a large number of rules and this is costly if it is done when every action is scheduled to execute. The solution is to invoke the specific ruleset for co-ordinating actions when one corresponding action has been scheduled.