Chapter 9

Scheduling Instructions

9.1 Introduction

The dynamics of a complex environment continuously instigates the agents to perform *instructions* (i.e., actions) in response to events. Some of these instructions belong to the animator's script and some are automatically generated by the C*ontroller*. Through a script, the animator can define a major set of actions, that is, actions that can be organised to take place at certain times and conditions in the animation sequence. Such actions serve as "landmarks", or act as a skeleton, around which new actions are instigated and fitted as opportunities arise. A script containing sparse instructions leads to a rather "unpredicted" or "simulated" animation in which the activities of the agents are mostly in response to stimuli exerted by the environment. In contrast, a script with detailed sequence of actions attain the behaviour of the agents of the prescribed times. The role of the animation system at the first stage is occupied with the scheduling of the scripted instructions and the building of the background actions that support or complement the scripted actions as they are developed.

Within a given animation environment, the motion of an agent can be affected in many ways. For example, one agent may have his attention diverted by another agent that has just entered the same area. An agent may perform an action in response to a request sent by another agent. An agent may have to temporarily, or in definitely, suspend a present goal because he has been instigated to perform a new and more important action. This is equivalent to saying that an agent may have more instructions to perform than he can cope with, so only the interesting ones are chosen. There is the case that one agent is crossing the path of another agent preventing him following the original path. This will, to some extent, affect his original behaviour. In the case that an agent is situated in a location marginal to the areas of activities and has become idle, he can perform an activity typical of his role such as looking around, having a drink, cleaning up, etc. If last action an agent was to pick up a glass, the agent will keep hold of the glass.

Therefore, due to the diversity of sources of stimuli acting upon the agents, a scheduling mechanism is required to ensure the selection of the most interesting instructions. Instructions are classified according to the kind of motions and to their origins, so that a systematic criterion can “distinguish” different nuances of the characters’ behaviours. As the scheduling of activities is strongly influenced by the allocation of resources, the task control type plays an important part. This issue has, to a certain extent, been approached in the previous chapter.

9.2 Overview of the Scheduling Stage

The Scheduling stage is the first and the highest component in the control hierarchy. The main goal of the scheduling procedure is to keep the agents busy with activities. It is also responsible for deciding when to initiate new instructions and for giving preference to the most appropriate ones. The scheduling of motions is actually achieved by two levels of control in combination: the Scheduling level itself and the Task level. On the Scheduling level, goal instructions are scheduled to run depending on the current load of the agents’ activities, that is, the agents are prevented from being unnecessarily overloaded with diverse actions. At the task level, where goal instructions are developed into partially expanded plans, decisions about the execution or suspension of an individual *task* of a process is performed. The suspension or termination of one process will consequently bring others into consideration. Normally such decisions are based on the availability of resources and the priorities of the contending processes. Basically this issue gives rise to the main aspects considered in the next sections which are: the category of the instructions, the sources of the instructions, the triggering of subsidiary instructions, and issues regarding priority for performing actions. The importance in identifying the different types of motions is to explore feasible combinations of motions which allow a more interesting and realistic animation. Figure 9-1 presents an overview of the types of motions managed in the scheduling scheme.

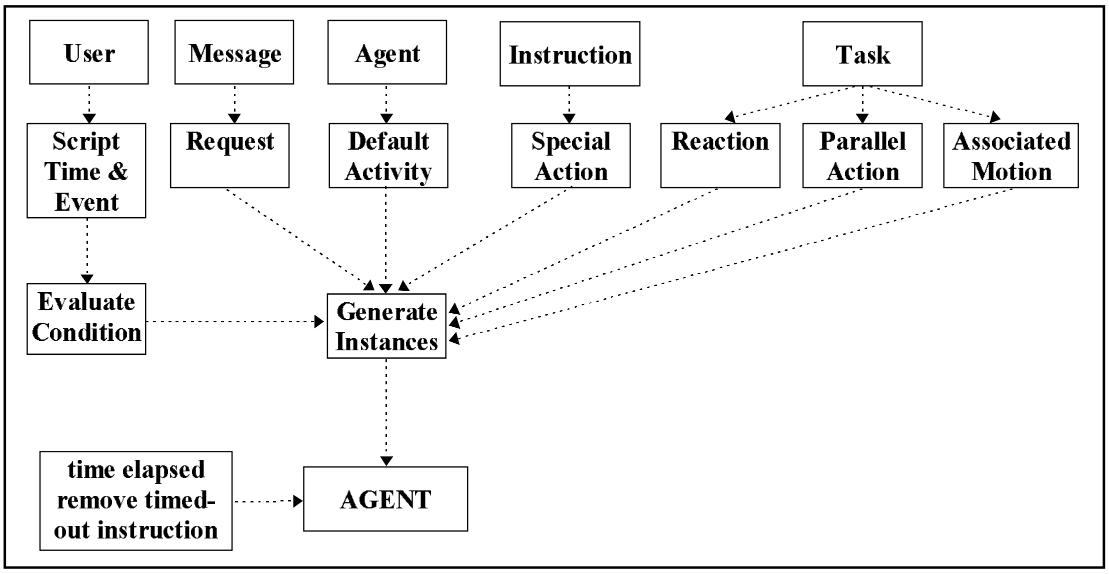

Figure 9-1: Overview of the Scheduler control.

9.3 Categories of Movements

Morawetz and Calvert have classified the movements of the human body as being primary or secondary depending on their nature [Mora90]. The primary movements correspond to actions which are performed purposively by the animated agent. The secondary movements, or "gestures", are minor actions which are usually developed unconsciously. They complement the primary movements giving them a more realistic appearance.

In our work we have adopted this classification and have extended it further in order to achieve a more lively animation. Therefore, we have the primary motions, which are typically characterised by the locomotion and the handling of objects; and we have the secondary motions which typically accompany the primary motions or that help with balancing the body. The third category of motion is the "signalling movement" which is a form of interpersonal communication, for example, an agent uses the arms to signal to another agent as a form interaction.

The categorisation of movements is applied not only to a specific motion, which is identified by a task type entity, but also to the overall process of a goal action. Therefore, both the task type and the instructions type are assigned to categories, this is based on the overall purpose they represent. This is particularly important at the task level where decisions are based on the original circumstances in which the instructions were created and not on each one in isolation. Obviously such a categorisation of instructions to types and the diversification depends on how the animator desires the instructions be scheduled. The purpose of this scheme will become clearer in following sections. Firstly we examine these categories of instructions.

9.3.1 Primary Instructions

Most of the actions realised by the agents are typically primary instructions whose purpose is to abstract specified goals. Normally two primary instructions cannot be performed simultaneously by an agent because their goals can be disparate. In the implementation terms, recalling that an instruction is ultimately developed into a sequence of tasks, it is impractical to interleave sequences of tasks especially if they need the same resources, handle different objects, or are performed in different places. Furthermore, the merging of two ongoing plans may lead to unexpected results by affecting the course of the plans.

9.3.2 Secondary Movements

Secondary movements are largely involuntary movements and the existence of which very often depends on the occurrence of specific movements represented by tasks. That is, certain tasks have associated secondary movements which are described by the slot *secondary* in its frame and these are triggered by the execution of the task. Furthermore, the secondary movements are terminated when the corresponding tasks have been terminated. Such secondary movements are themselves represented by instructions, or by tasks in more specific roles. As their purpose is to accompanying other movements, secondary movements are the least importance of all motions and therefore they are susceptible to interruption by motions that require their resources.

Nevertheless, the applicability of the secondary movement can be extended to other situations such as a complementary motion which is discussed in the next section. For example, in the locomotion activity, in addition to swinging arms, automatic straightening of the torso is also a desirable feature especially when the previous action has left the torso out of its normal position.

The parallelism of motions is not a scheme exclusive to the secondary movements as described in this section. It is also useful to simulate parallel motions that are not specifically secondary to an action, nor it is a main action specified in a plan. Typically a plan is linear and parallel actions cannot be specified. Such motions are independent activities and they are triggered in a similar fashion as those of secondary movements. Thus, this helps to overcome the problem of linearity of the plans. An example of this is the action of walking while holding an object (e.g., a glass), after having picked it up. Because this particular example is a relatively complex problem that represents a general situation, the mechanism of triggering actions with associated motions will be discussed later in these chapter.

9.3.3 Signalling Movements

The movements for signalling share some features common to the other two groups of movements. The movement of an agent waving a hand to another in acknowledgement or the movement of an agent lifting his arm to get the attention of a *barman* are examples of signal movements. These movements are actions that implement the communication between agents. These actions in their turn are represented by plans that combines both a (gesture) movement and the *message* mechanism discussed in Chapter 7. Signalling movements can also be seen as being in the secondary movement class partially because they employ motions that are optional or not so important, typically gestures. However, depending on the context for which the plan has been selected, the motion performed may also require turning the torso, or even changing the posture of the body to face the target, in order to signal properly. Nevertheless, this category of movements is normally of a short duration and for a specific purpose, which minimises the time a motion is interrupted.

9.4 Sources of Instructions

Instructions leading to a goal executed during an animation originate from different sources. The two main sources are the animator's script and the animation system. Because the source of instruction is an important criterion in scheduling the animated activities, the animation system as a source of instructions allows a further classification into at least four types of sources as shown in Figure 9-2. The information regarding the source of the goal instruction is stored at the *source* slot in the *root* instance.

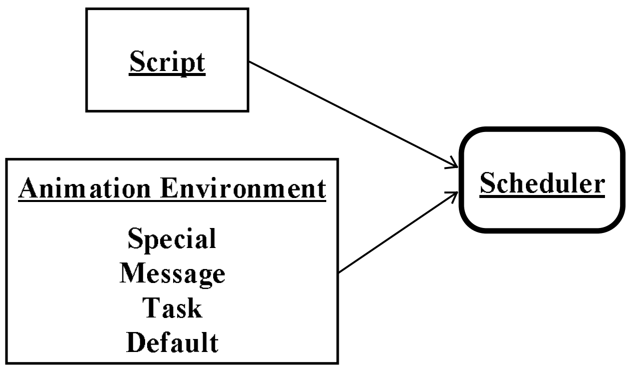

Figure 9-2: Source of activities.

9.4.1 The Animation Script

The activities to be realised in an animation are constituted as a collection of instructions which can be organised by the animator as a script. Such instructions are prescribed to happen at established times or under defined conditions. These instructions define a main line of actions which act as a skeleton, or a rough sketch, of an animation plan. They are expanded during the animation. It is not unusual for an animation script be composed of up to five main actions, having many more gaps than prescribed actions within the total animation period. Additionally, through scripts the animator can introduce auxiliary commands to alter the configuration of the environment or the attributes of the animated objects, and these are facts that may be considered in the planning process. Through the use of scripts the animator can have the control of the animation thus the script, as the source of instruction, have precedence over those of the system. The general format of a script instruction is given by:

agenda(*<instr. number>*, *<Condition>*, *<Activity>* **)**

<Condition> gives information about the time at which the instruction is due to occur, some examples are:

***time*(** *<number>* **) -**	the time the instruction is to due start;
***event*(** *<Clause>* **) -**	when the fact defined by *<Clause>* occurs;
***followup*(** *<instr. number>* **) -**	a condition to start is given when the instruction specified by *<instr. number>* terminates;

<*Activity*> is the action to be performed by the agent or a command to create a new object.

Examples:

```
script( 1, time(1),  create(john, customer, [attr(colour,blue), attr(place,door_a)]) ),
script( 1,  time(1),  action(john, sit_i, []) ) ),
script( 2,  event(clause(john, posture(sit))),  action(mary, meet, [john] ) ) ),
script( 3,  followup(2),  action(john, call, [supplier] ) ) ),
script( 4,  time(100),  action( peter, sit, [green_chair] ) ) ) .
```

9.4.2 Actions Triggered by the System

The automatic generation of actions by the animation system extends those actions planned in the script. Three kinds of sources of actions are described in this section, however, new variants of sources might be identified by the animator which may provide the animation with more versatile alternatives. New actions can be initiated in any point of the process (task, instruction, or message) by asserting special clauses onto the blackboard. These clauses are recognised and processed by the Scheduler generating instructions which may potentially be performed by the agents as opportunities arise.

9.4.2.1 Actions requested through Messages

The Message is a special means by which agents can communicate. In fact, the communication is limited to requests for actions and acknowledgements. That is, the message is an entity which provides the carrier for encoding an instruction and this instruction is scheduled to be undertaken by an executor agent, which is the receiver of the message. In Chapter 7, a scheme was proposed using two complementary types of messages implementing the communication between two agents: the *request* message and the *acknowledgement* message. The *request* message sends typically a request for primary motion to the executor while the *acknowledgement* message sends a request for secondary motion (signal) to the first agent. Because such actions are instigated intentionally by agents and because more than one agent is involved, the message as source of instruction is regarded as higher priority for scheduling than most of other sources in the animation system.

9.4.2.2 Task Generated Actions

The task level, as a source, accounts for most of the actions generated by the system. Actions are triggered at the task level in different ways for a number of purposes. These methods of triggering have been presented in the previous chapter, however, they are summarised here in order to present the overall scheduling priority rating.

- the associated action, or accompanying motion. This is typically a secondary motion which is triggered along with the main motion within a task procedure. This is done irrespective of whether it will ever have a chance to gain access to the resources. The associated actions have the lowest priority rating and consequently they may not be activated if the required resources have previously been committed. In the case of a locomotion the task of swinging arms is always launched. The accompanying motions can also be called as co-ordinated actions as they operate in combination with the main motion. For example the left arm swings forward whenever the right leg is swinging forward.

- the preliminary action, or special action. When a specific condition is not satisfied for the realisation of a motion then the task fails and the control is handed back to the parent instance. Eventually the parent instance will realise that none of the existing plans is acceptable except the *alternative instruction* specified in the *alternative* slot. Such an *alternative instruction* will work as a preliminary action which activity changes the current condition to the condition required by that the failing task. Normally such a preliminary action is required as a result of one action being interrupted by another, this affects the condition of the agent which cannot proceed with the action without a preparatory operation. For example, a *waiter*, who is performing *glass collection*, has both arms busy with glasses and a *customer orders a drink*. In order that the *waiter* can properly perform the customer's request he fails to pick up more glasses, and executes the alternative action which is to *dispose of glasses*. Such an alternative action is a special one which has the highest priority of execution and therefore any other task will only be considered by the agent when the special action is completed.

- the continuation action. This is a useful way of providing a backup action that can give continuity to the current motion. The continuation action is triggered in the conclusion of a step of motion which mechanism has been presented in section 8.7.3. One example is to schedule an action to turn the torso back to straight direction after having turned it to one of the sides. The other example of this continuation of actions is holding a glass after having picked it up. Because the continuation action is generated independently of any other action or plan, it is called *parallel action,* or *parallel task* in a more specific activity. This example is shown in the next section.

- the reaction action. This is done in a similar manner to the "continuation action". That is, a special clause (*broadcast*) is asserted onto the blackboard during the execution of a task. The Scheduler recognises the clause which is aimed at creating responsive behaviours from other agents in the area. For example, during the walking task, at the completion of each step, the agent's database is updated with facts about the new state (e.g., John's support leg changes from the right leg to the left leg), location, direction, etc. Suppose the situation given in Figure 9-3 that *John* has crossed into a new region, additional information is updated in the involved region's database (*John left the bar area* and *John entered the restaurant area*) and a special clause is thus placed on the blackboard with the purpose of "calling everyone's attention to the new area". This clause causes the scheduler to schedule reaction instructions to the surrounding agents (e.g., *Mary* and *Waiter*). Obviously, the specific reaction instruction depends on the animated scenario and the specific agents. In this example, *customer agents* may just glance at a newcomer, while the *barman* could offer his services to the newcomer at the next opportunity, and the *waiter* could give him an acknowledgement.

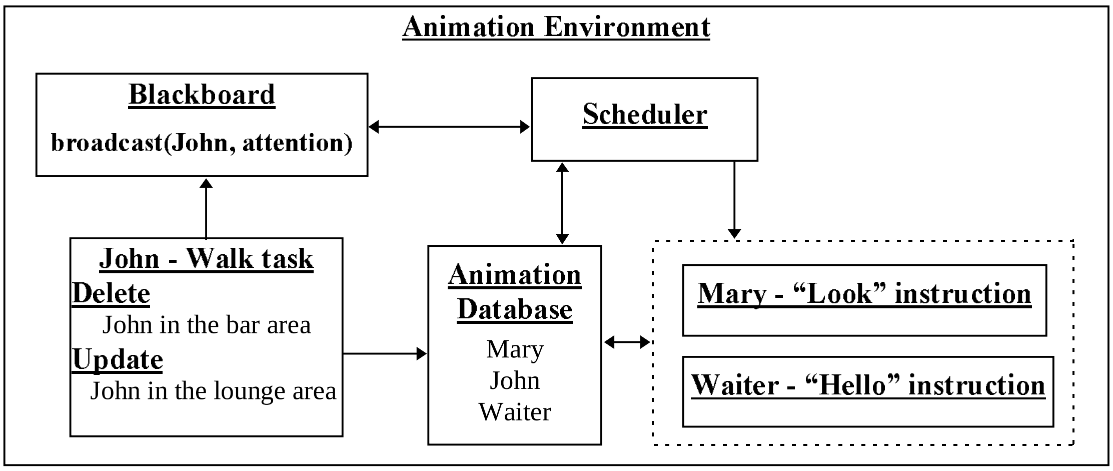

Figure 9-3: The event "John entering a new area" causes reactions from others.

9.4.2.3 Default Motion

In the complete absence of activities from a given agent, the scheduler starts the default instruction with activities typical to the agent. Such an instruction is specified in the *activity* slot of the agent frame. The Scheduler realises that an agent has become inactive because a special clause (*idle_agent*) has been placed on the blackboard by the *end* state of the last instruction performed by the agent. The default instruction is in fact a general one that groups a number of main actions which are typical of that kind of agent (*barman*, *waiter*, *customer*, etc.). The selection of each of these actions depends on different conditions, for example, whether the instruction has not been performed by the agent in the last 100 frames or has not been performed as the last instruction, etc. That is, the main purpose of this condition testing is to avoid repetition of one of the default actions.

9.5 Scheduling of Priorities

In dynamic environments agents are subject to many influences that can affect their plans or even change their goals. Such environments allow a diversity of interactions among agents which can be developed to generate a more realistic impression of the animation, however, this diversity very often gives rise to conflicting actions where more than one task claims the control of the same resources for different purposes. In such a situation the scheduling mechanism is required to select the relevant actions, terminate the irrelevant ones, or make a conciliation between them. This can be achieved by establishing the concept of priority and using it to arbitrate in the scheduling of instructions. The sources discussed in the previous chapter are organised in the order of priority shown in Figure 9-4.

Special
Script
Message
Continuation
Reaction
Default
Associated

Figure 9-4: Priority according to source of action.

In the previous sections animation instructions have been characterised in terms of categories and sources. These serve as criteria for differentiating actions. The scheduling process in the scheduler level is simple: all actions are eligible to start. However, the issue is whether an action will really start or which will start first. In the case that an action does not start immediately it is included in the *callback* list of the executing process. The criteria are:

- A signal action will interrupt a primary action irrespective of its origin. As signal actions are of very short duration, in most cases lasting no more than one *step* of motion, they do not disturb the primary actions too much.

- If the category of the new action is the same of the current action. Two possibilities occur:

a) If the new action has a higher priority than the current one, the currently executing action is set to the interrupt condition. The interruption will take place when the current *step* of the task has been completed by the BAS, the pending process then re-evaluated and activated.

b) If the new action has less or equal higher priority than the current one, then the new action is simply included in the *callback* list of the current process and it is only called back when the current process has been completed.

- If both the new action and the current action are secondary motions, the priority test is based on the source of action as in the previous case. However, if the secondary motion is interrupted by one of higher priority it is not called back. Similarly if the new secondary motion has lower priority it is simply ignored and discarded. This purging of a secondary motion does not cause any problems since the main task will, if required, start a new associated motion as each *step* is completed.

9.6 Timing-Out Processes

A goal action may not always be developed into an active process in the same frame time as it is requested to execute. However, if the action is not performed within a few frames and if, for example it is a reaction towards somebody's presence, it may have become meaningless and is therefore discarded. The criteria for timing out actions is mostly based on their origin, that is, on their degree of importance, with each source having a different associated time-out. In our approach, once an action has started within the allowed time, the time-out is reset to a large value so that it no longer affects the action.

9.7 Example of "Forking" *Parallel Actions*

The example in this section is intended to illustrate the operation of the "continuation action" that is generated by some tasks, such a scheme has been presented in the section 8.7.3. Here we take the opportunity of summarising some of the aspects involved in motion coordination in a particular example. We have argued previously that an agent cannot perform two primary actions simultaneously and this still holds. However, there are situations in which an agent needs to perform two actions simultaneously towards a single purpose. For example, the goal of collecting used glasses, is an action described by the transfer of a glass (or two glasses) from tables to a counter. Generally speaking, in this process the plan of the agent comprises a sequence of actions depicted in the Figure 9-5a, namely, *pick up the glass from the table*, *walk to the counter*, and *put the glass on the counter*. This part of the plan is simple and appears to be a sound one to the animator, however, it is incomplete. If we literally visualise the plan as being executed by the agent we would actually see him picking up the glass and walking to the counter with the glass swinging along with one of his arms as if the glass were an extension of the limb. Clearly this situation occurs because there is no explicit action in the plan saying to *hold the glass up and stable*. Nevertheless, if an action of *hold the glass* were inserted in some point of the above plan, as shown in the Figure 9-5b, the agent would simply hold the glass only for the duration of the *hold glass* action and ignore the existence of the glass as soon as the plan advances to the next sub-goal.

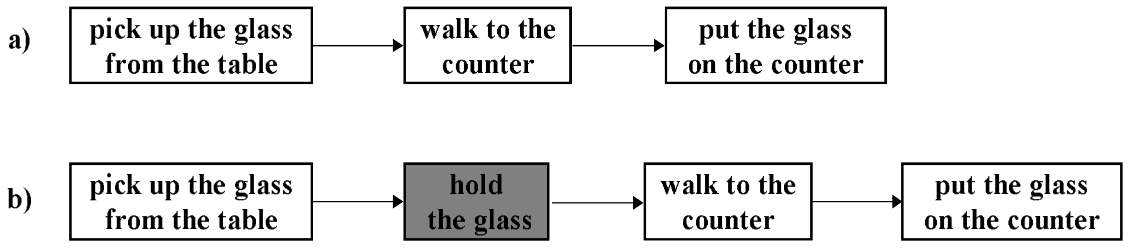
a)
pick up the glass from the table
walk to the counter
put the glass on the counter
b)
pick up the glass from the table
hold the glass
walk to the counter
put the glass on the counter

Figure 9-5: Two versions of plans for the "collect glass" action.

The sequential nature of a plan can be overcome by creating a *parallel task* which can be activated if the resources are available and run in parallel with the current action. This kind of task is an activity undertaken independently of the ongoing actions. In the example, the activity of *hold glass* is started as soon as the agent gets hold of the glass, that is, after having achieved the action "pick up the glass from the table". Thus, such a task cannot be defined as part of a plan especially if it may run indefinitely. In fact the duration of the *parallel task* may extend beyond the original task that has started it, or even beyond the duration of the plan. For example, if the goal of an agent is to get a drink then the agent picks up the glass and it is done. The *hold glass* task is started immediately afterwards and it remains active while that particular holding arm is not required for any other activity. The following aspects are observed in the handling of a *parallel task* and they are discussed in more details in the next sections:

- the *parallel task* is automatically started by an executing task and it gains activity if no higher rated actions occur;
- if the *parallel task* handles an object it "locks" the access to the resources as it comes to action;
- the *parallel task* is also allowed to continue by a subsequent task, that is, it runs in parallel with other actions;
- the *parallel task* may act as an intermediate motion connecting two other motions.

However, if eventually a task takes over of the resources used by the active *parallel task*, it will take into account the condition that the resource (the arm) is in and give the proper continuity to the current motion.

9.7.1 Motion Continuation

In the example of the action of *hold glass*, the execution of this activity causes the agent's arm reach the *holding* position. Once the agent has approached the counter, the action of *put the glass down* expects the arm to be in holding position so that the *put* motion can be performed. Therefore, continuity not only requires that resources be allocated but also the checking of their current position or state. This is important as the potential continuation tasks may use the same resources for different purposes. Obviously, the lack of verification of the state of the resource may lead to an incoherent continuation. Internally, the state of the limb holds the information of the kind of action being carried out. For example, the action of *waving the arm* does not follow the one of *holding of a glass*, however, the action of *cheering with a glass* follows properly. The example of Figure 9-6 show a *parallel task* being continued by a task within the same plan. In Figure 9-7 the *parallel task* is launched as the original plan is ended, and eventually it is continued by a *cheer* action.

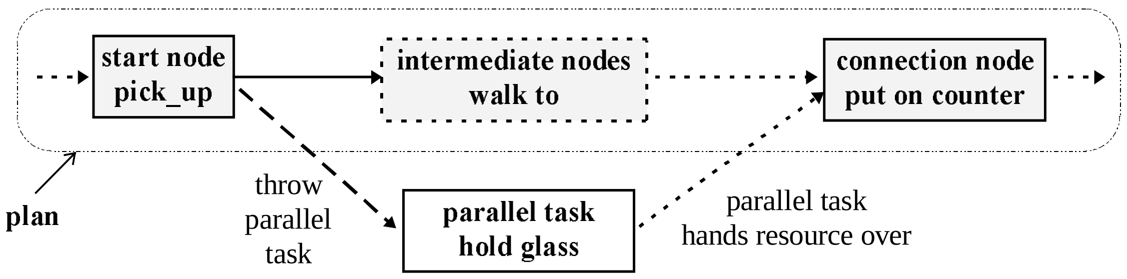

Figure 9-6: P*arallel task* hold glass is initiated and terminated within the same plan

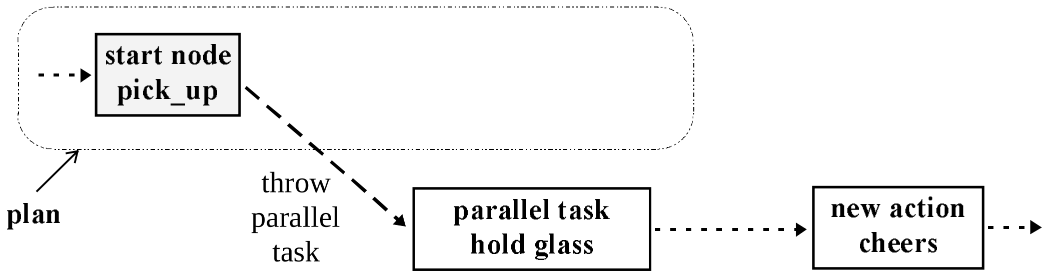

Figure 9-7: The action of cheering occurs opportunistic while the agent is holding a glass.

9.7.2 Intermediate Motions

When the successor of a *parallel task* does not find the required resource in the expected condition, an intermediate action is thus started in order to bridge the current motion (the *parallel task*) and the following motion (the *successor task*). The intermediate action is an instruction that brings the resources from the current state to the expected state. Figure 9-8b depicts this process schematically as a connection that forms a sequence of three successive motions.

Very often the *parallel tasks* require resources to handle objects. In such cases where objects are involved, the allocated resources are annotated in the agent's "*lock list*" slot. If the required resources are indicated in the *lock list* they can be re-allocated provided they satisfy both the priority and *motion continuation* criteria (Figure 9-8a). In the case of a lack of continuity the intermediate action will attempt to bridge both tasks. That is, if in our example the agent wishes to wave to somebody he would put the glass aside, as a special action, and start waving (Figure 9-8b). The assignment of the special instruction to the highest priority avoids unnecessary complexity in the control of the involved tasks. At the end of the special action the new instruction, *wave arm*, is thus re-activated if no higher priority action occurs before its start.

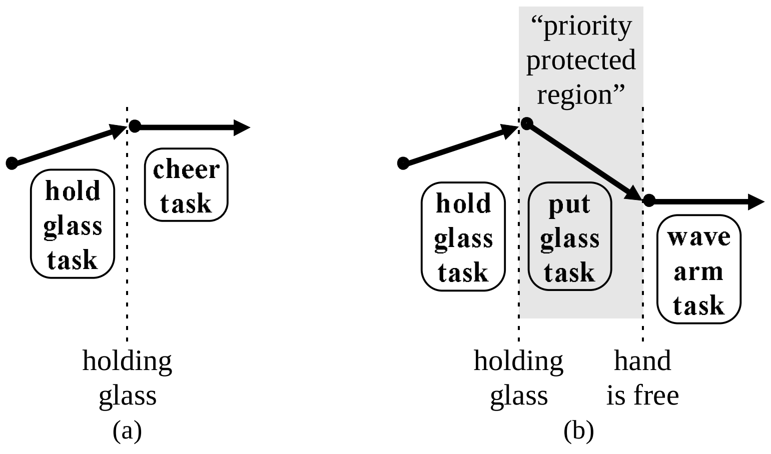

Figure 9-8: Continuous and interrupted sequences of motions.

9.7.3 Coordinating Multiple Interleaved Actions

The *parallel task* feature is a convenient way to automatically generate actions that are expected to occur without having to specify them in a plan. It is realised opportunistically in such a way that it blends in a natural manner with the executing actions as if it were part of the plan that has triggered it. Figure 9-10 presents a set of rules that implements the *drink* action and we consider the following scenario to execute it: an agent is sat down at a table and he has received a glass which is on the table; then the agent is about to execute the drink action as part of a major plan. Additionally, we suppose also that the amount in the glass allows three sips. The execution of this plan will apparently generate two threads of actions: the drink action and the hold glass action. The drink action is depicted in the left column of Figure 9-9. It basically comprises the whole activity performed by the agent, except the *hold glass* action. The *hold glass* action is not a single thread, but actually two instances of the *parallel task* which are started by the *drink_t* task (i.e., the *have a sip* box). Initially the *hold glass* is started by the *pick up* task but it is not activated because *drink_t* task immediately follows *pick up* task in the plan and has higher priority. After the executing of *drink_t* task another *hold glass* is launched and this time it is activated because no other task requires the holding arm as a resource. The overall motion execution in Figure 9-9 is thus denoted by shaded rectangles which are the tasks actually executed.

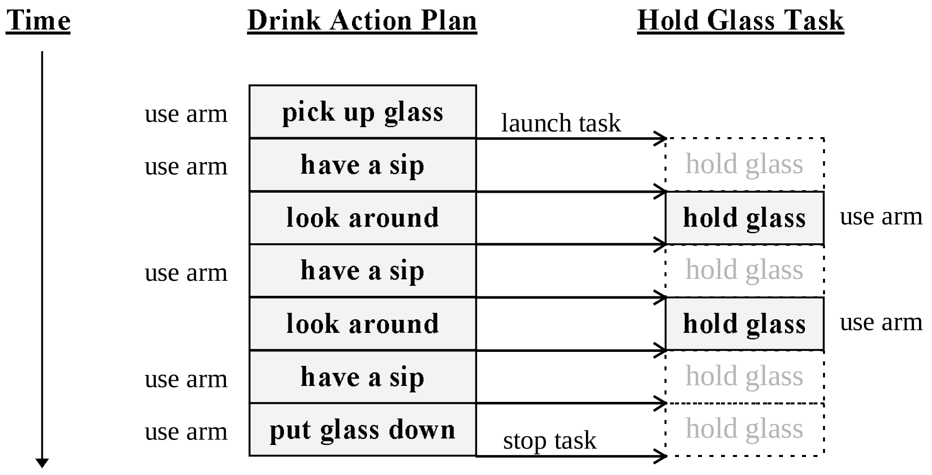

Figure 9-9: Example of parallel tasks developed by the drink instruction.

```
rule end_drink_on_hand
if  the Agent performs drink_i using Instance and
   Agent drink_has_finished and
   Agent has_drink_on_hand
then
   choose_plan( [[put_i, place, drink, drink_hand]] ) .

rule end_drink
if  the Agent performs drink_i using Instance and
   Agent drink_has_finished
then
   done( Instance, drink_11 ) .

rule have_a_sip
if  the Agent performs drink_i using Instance and
   Agent had_no drink_i since 30 frames ago and
   Agent has_drink_on_hand
then
   choose_plan( [drink_t , drink_i] ) .

rule pick_up_the_drink
if the Agent performs drink_i using Instance and
   Agent has_drink_on_the_table
then
   choose_plan( [[pick_up_i, table, drink, condition, drink_hand],
                        straight_up_i, drink_t, drink_i] ) .

rule have_a_look_around
if the Agent performs drink_i using Instance and
   Agent look_at_people_around
then
   choose_plan( [look_around_i, drink_i] ) .
```

Figure 9-10: Set of rules implementing the drink instruction.

9.7.4 Conclusions

The homogeneous blending of the *parallel task* with an existing scheme permits a coherent development of the agent’s behaviour. The multitude of plans for the different behaviours that the agents can perform are not restricted because the framework was conceived to be multi-process environment. As such, new solutions can be smoothly incorporated into the framework. The example of the agent enjoying a drink is not restricted to pick-drink-put but it can be as natural as pick-drink-look_around-drink-put.

9.8 Summary

The source of instructions is divided into two main groups: the script and the animation environment. In normal circumstances the script has the highest priority because it has been defined by the animator. The instructions originating from the environment are further divided into "sub-sources" in order to associate with them a finer degree of priority. In order to identify different combinations of motions, actions are differentiated as being primary actions, secondary actions, or signals The purpose of the Scheduler is to co-ordinate the activities of the agents and schedule default activities when the agents are idle. The purpose of the Scheduler is wide and open to further implementation that can add liveliness to the animation.