EXAMINING THE GENERALITY OF A BEHAVIOURAL ANIMATION FRAMEWORK

Hanna J.R.P., Millar R.J., Johnston W.M.

School of Computing and Mathematical Sciences
University of Ulster, Shore Road
BT37 0QB Newtownabbey
United Kingdom
p.hanna@ulst.ac.uk http://www.ulst.ac.uk

ABSTRACT

This paper tests the hypothesis that it is possible to build a generic Behavioural Animation system. The paper outlines the four models that form the key elements of a Behavioural Animation system, and a proposed generic structure for the Behavioural Component. Three implementations, all of which follow this structure but which differ in implementation detail, are then presented. Each implementation is tested with several animations and the results tabulated. It is then shown that the previously proposed structure is sufficiently generic if goals are deleted after one cycle. It is also demonstrated that, if goals are to remain in the goal queue for more than one cycle, a rule-based system for deciding when to delete a goal is required in order to achieve the required level of genericity.

Keywords: animation, behavioural animation.

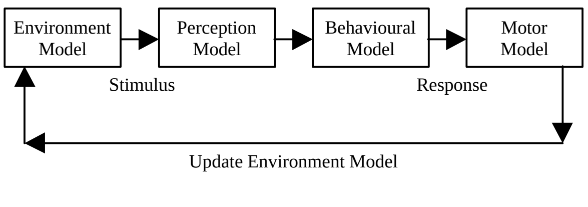
1. INTRODUCTION

This paper is based upon the work of Millar et al [Milla99a] who proposed that all Behavioural Animation systems can be modelled by a common framework. This structure is given at its most abstract level in Fig. 1, which shows the four interacting models that comprise the system. A more detailed framework for the Behavioural Component was also proposed in the original work. This paper presents the results of an experiment whereby three variations of the Behavioural Components were implemented. All conformed to the framework proposed, differing only in implementation detail. Each was then tested with several different animations. The results of these tests are then used to evaluate the level of genericity offered by each implementation.

2. BACKGROUND

2.1 The Original Hypothesis

Millar et al. [Milla99a] presented a review of Behavioural Animation in which they investigated the characteristics of Behavioural Animation systems produced by various researchers. They found evidence that all Behavioural Animation systems could be characterised by a number of key concepts. Based on this, they proposed that it was possible to build a generic Behavioural Animation system that would be capable of producing a wide range of animations with no modifications to the source code. The benefits of such a generic system for both research and industry are clear.



Framework of a generic Behavioural Animation system as proposed by Millar et al [Milla99a].
Figure 1

In their paper, Millar et al. analysed the various tasks undertaken by a Behavioural Animation system and presented what they found to be the basic groupings of functionality that are common to every such system. By separating out the functionality in this manner, they arrived at a set of four models, each of which modelled a cohesive aspect of the animation process [Blumb95a].

The *Environment Component* would maintain information on the virtual world and all the entities within it. This includes not only information such as the position and appearance of each entity, but also the internal state of each entity, for example hunger levels or other internal variables. Thus, the Environment Component stores both the 'physical' and 'mental' state of each entity.

Most Behavioural Animation systems provide incomplete knowledge of the virtual world to the entities perceiving it [Reyno87a, Renau90a]. The *Perception Component* provides a common way of specifying the sensory limitations of entities within the virtual world. Each entity is likely to have its own Perception Component. For example, an insect may be able to detect the smell of a flower at a longer distance than a human. This example also illustrates that the Perception Component is not limited to simulating sight: it can simulate any other senses that are required by the animator.

The *Behavioural Component* models how each entity responds to what it is sensing via its Perception Component [Reyno87a]. Each entity has its own Behavioural Component, giving the animator the essential ability to assign different behaviour to different entities. In the past, the Behavioural Component has been hard-coded. However, Millar et al. proposed that the animator could use a common 'language' to define behaviour, thus negating the need to modify and recompile the source code each time a change in behaviour is needed. This would greatly increase the productivity of the animator. The Behavioural Component decides courses of action and outputs these to the Motor Component.

Just as an entity does not usually have the ability to sense everything in its virtual world, so it is also usually incapable of unrestricted movement within its virtual world. The *Motor Component* exists to limit the actions of entities by, for example, limiting their speed of movement. It provides a set of movement primitives that may be invoked by the Behavioural Component [Reyno87a, Takeu92a]. As well as serving this purpose, the Motor Component also serves to model details of the entity's movement. Good examples of this include modelling the wing flaps of a bird or the leg movements of a human.

A fifth component that can be added, although it is not strictly part of the model, is *the User Intervention Module*. This module permits the animator to intervene in an animation and modify the physical or mental attributes of entities and the virtual world.

The interaction between these components is summarised in Fig 1.

2.2 The Proposed Generic Model

After having presented these findings, Millar et al. proposed a structure that could be used to implement a generic Behavioural Animation system based on these four models. Central to this structure was the Behavioural Component, which would form the core of the system and which would be the controlling component of the system.

The Perception Component and Motor Component would communicate with the Behavioural Component via standard interfaces. This level of modularity would permit different models to be arranged together without needing to make any changes to the source code. The information passed between the Perception Component and the Behavioural Component would be details of detected nearby entities, but would not need to specify whether the information was gathered using the sense of sight, sound or smell etc. Similarly, the Motor Component would be given instructions on what the entity was trying to do, but it would be a matter internal to the Motor Component to decide whether this resulted in, for example, flight, walking or swimming.

All the entities and their components are stored within the Environment Component. The Behavioural Component, which is duplicated for each entity in the system, itself consists of four main modules:

2.2.1 The State Variables

This part of the Behavioural Component stores the physical and mental state of the entity. The physical state is recorded by storing information on the entity's position, appearance and perceived surroundings. The mental state variables are analogous to the memory of a real creature. The animator is free to add any desired variables to this area. For example, they may create a 'hunger' variable with which to more accurately model a creature hunting for food. Another example may be the level of 'fear'.

Also within the state variables module are 'mental triggers'. These are rules that are fired at regular intervals to modify a specific variable. For example, an animator may create a mental trigger to steadily increase the hunger level of a creature over time.

2.2.2 The Rule Base

The rule base consists of a series of rules entered by the animator using a common language. Each rule takes the form 'if this set of circumstances is met, then carry out this action'. When evaluated, the Boolean value of each rule will be determined. Each rule that evaluates to 'true' will establish a goal that will be placed in a queue private to that entity. Each goal also has a priority level and a lifetime. The priority level will allow some rules to be deemed more important than others. For example, fleeing from a predator is usually more urgent than eating food. The highest priority goal in the queue is taken and executed for one cycle. The lifetime allows goals to be removed from the queue after an interval of time, to prevent decisions made in the past affecting behaviour under changed circumstances.

2.2.3 The Memory Module

The memory module consists of the aforementioned goal queue. The goal queue allows an entity to recall activities that it intends to do. For example, if an entity successfully evades a predator, it can recall that it had intended to eat some food. The memory module is also responsible for deleting goals that have outlived their lifetime.

2.2.4 The Movement Module

The movement module takes goals that are to be carried out, and translates these into a series of generic movement commands that are to be transmitted to the Motor Component. The movement module is also responsible for path planning and collision avoidance.

2.3 Testing the Hypothesis

With the goal of testing the hypothesis that a generic Behavioural Animation system can be produced based on the structure outlined by Millar et al., a Behavioural Component was implemented following this structure. As the Motor and Perception components were secondary to the Behavioural Component in the proposed structure, so the implementation concentrated on the Behavioural Component. A Motor Component and a Perception Component were produced with simple functionality, with the purpose of supporting the Behavioural Component. The Perception Component was implemented with a 'perception range', whereby entities can sense other entities within a specific range, and cannot sense entities beyond that range. A simple User Intervention Module was also added to give the animator greater control.

Although the structure proposed by Millar et al. was detailed, less rigid details were given as to the manner in which the associated operations should be implemented. Therefore, several implementations were made which were similar on the macro scale, but which had key differences in implementation detail.

3 IMPLEMENTATION APPROACHES

3.1 Common Implementation Details

The implementation follows the discrete frame-based approach common to many types of animation [Thoma84a]. An integer frame counter provides the 'time' element of the animation. This timer is incremented by 1 each time a frame is generated. All time intervals are therefore expressed in frames.

Each entity is represented by an Object-Oriented class, which encapsulates the data structures outlined by Millar et al. The rules are expressed in a simple language developed for the system. The language allows the animator to query the environment and to combine such queries in a boolean manner. When executed, these goals trigger interaction between the Behavioural Component and the Perception Component.

In addition, the language allows the related goals to be expressed in a common format. Each goal represents a possible course of action. The system supports actions such as 'flee from entity', 'go to entity', 'fight entity', 'kill entity' and 'wander'. The 'wander' goal causes the entity to move in a random manner and is intended for use whenever there are no other actions to be performed. Whenever a goal is created, additional information is attached to it. For example, in the case of 'go to', the target and preferred speed of movement are attached.

In order to enhance functionality, each goal has a number of associated further instructions. These instructions allow further changes to be made when a goal is actually executed during one frame. For example, when an entity eats food, a further instruction could be attached to that goal which resets the 'hunger' level to zero. A goal can have zero or more further instructions.

In each implementation, the system carries out a complete set of actions each time a frame is generated. Once each frame is generated, it is displayed on screen. This process is repeated for each frame. Between frames, the animator has the opportunity to intervene and modify the state of the animation. Note that in each implementation, the animator can specify the order in which entities are parsed, and this order can be changed as the animation progresses.

3.2 Implementation 1.

The actions that are performed with each frame of the first implementation can be summarised as follows:

1. The system executes all the mental triggers for each entity. For example, if there is a mental trigger to increase hunger level, the hunger level variable will be incremented at this point.
2. The system executes all the rules for each entity. This involves interaction between the Behavioural Component and the Perception Component. For each rule that evaluates to 'true', the associated goal is added to the entity's personal goal queue. Each goal has an associated priority level and a lifetime. The lifetime is expressed in frames.
3. The system chooses and removes the top priority goal from the goal queue. This goal is then executed for one frame. This involves interaction between the Behavioural Component and the Motor Component. Note that if there is more than one goal with the highest priority, one is chosen at random. Additionally, if there are no goals in the goal queue, then the entity performs no action during that frame.
4. The system goes through the goal queue of each entity and deletes all expired goals. Expired goals are goals that have been in the goal queue for longer than their specified lifetime. Since goals are removed after being executed for one frame, expired goals are always goals that have not been chosen for execution.
5. The system displays the state of the virtual world on the screen and invites the animator to intervene. Assuming that the user has not chosen to terminate the animation, control then returns to action 1.

The key features of this implementation are thus:

- Goals are added regardless of whether the same goal was added on a previous frame. It is thus possible to have duplicate goals in the goal queue. This is necessary since an executed goal will be deleted after one cycle, even if it has only partially met its goal.
- Goals are deleted as soon as they have been executed for one cycle. The same goal cannot be executed more than once.
- Expired goals are purged.

3.3 Implementation 2.

The second implementation is an extension of the first. Mental triggers and rules are executed as before. However, the implementation of goals is different.

Instead of deleting goals after being executed for one frame, goals are deleted only when they have been 'achieved'. When a goal is executed for one frame, it returns with one of two values: 'achieved' or 'not yet achieved'. For example, the 'go to' goal is deemed to have been achieved when the entity is within a certain minimum distance from the target. Similarly, the 'flee from' goal is deemed to have been achieved once the predator is no longer within perception range. If the goal has been achieved, then it is deleted. If it has not yet been deleted, it remains in the goal queue.

Note that the 'wander' goal is never 'achieved' and will thus remain in the goal queue as a low-priority goal to be executed whenever nothing more interesting is to be done.

As the same goal can remain in the goal queue and be executed for several frames, it becomes unnecessary to add the same goal to the goal queue multiple times. Therefore, in this implementation, a goal is only added to the goal queue if it is not already present.

Finally, since goals are deleted once achieved, it becomes unnecessary to have a mechanism for purging goals. Therefore in this implementation, there is no facility for purging expired goals.

The key features of this implementation are thus:

- Goals are only added to the goal queue if not already present.
- Goals are deleted only when they have been completely achieved.

3.4 Implementation 3.

The third implementation is an extension of the second. In this implementation it is recognised that goals may 'fail' as well as being 'achieved' or 'not achieved'. As an example, if an entity is trying to 'go to' another entity but the other entity moves outside perception range, than the 'go to' goal has failed because there is no longer any information with which to continue the action.

Therefore, in this implementation, a goal returns either 'achieved', 'failed' or 'neither' when executed. A goal is deleted from the goal queue whenever it is either achieved or has failed.

As in implementation 2, not all goals can return all three results. The 'wander' goal cannot either be achieved or failed. Similarly, the 'flee from' goal cannot fail. If the enemy is within perception range, then the entity is still attempting to flee from it, so it returns 'neither' achieved nor failed. If the enemy is not within perception range, then the goal has been achieved.

The key features of this implementation are thus:

- Goals are only added to the goal queue if not already present.
- Goals are deleted only when they have either been completely achieved or have failed.

4 TEST CASES

In order to test the genericity of the system, a number of test cases were chosen. Each was chosen to test a certain aspect of the system and all were chosen to be as different from each other as possible, in order to test as much of the spectrum of animations as possible.

The first test case is a simple predator/prey situation. At the start of the animation, the prey and predator are within perception range of each other. The prey has the advantage of being able to move faster than the predator. Thus, assuming the entities have been programmed to follow the expected predator/prey behaviour, the prey will have moved beyond the perception range of the predator after a period of time. This test case provides an indication of the manner in which the rules governing the predator's behaviour are operating.

The second test case is a set of four entities with identical attributes whose behavioural rules cause them to engage and 'fight' one another. Winning a fight results in the losing entity being removed from the animation. After a period of time, only one entity will remain. This test case provides an illustration of three competing rules operating together – 'wander', 'go to', and 'fight'. These three rules have definite priorities: for example, 'go to' is always a higher priority goal than 'wander'.

In the third test case, an entity exists in a world that contains several pieces of food. The entity maintains an internal hunger level that increases with time in a linear fashion. It wanders randomly, ignoring the food when its hunger level is below a certain value. Once the hunger level reaches this certain value, the entity looks for food. Once food is found, the food is removed from the animation and the hunger level reset to zero. If the hunger level reaches a third, higher, value then the entity itself perishes. This test case provides an illustration of rules being dependent on an internally controlled condition.

The final test case is an implementation of zones of tolerance. An entity is initially placed at some distance from a disinterested creature that is moving randomly. The entity also moves randomly. However, if it strays within a certain distance of the central creature, a 'flee from' goal is triggered to move it away again. If it moves too far away from the central creature, a 'go to' goal is triggered that moves it closer again. The effect is for the entity to move within a fixed range of distances from the central creature. This test case demonstrates two opposing actions that are constantly competing with one another. Unlike the second test case, the two actions do not have definite pre-definable priorities over one another.

Table 1 summarises the behaviour as observed in each test case for each of the three implementations.

5 EVALUATION

In 9 of the 12 test cases, the animation proceeded as expected. However, there were two distinct cases when this was not the case.

5.1 Limitations of having no 'failed'

In the predator/prey animation under implementation 2, the predator was able to continue pursuing the prey despite the fact that the prey was beyond perception range. In this case a goal instructing the predator to 'go to' the prey had been added to the predator's goal queue. Each time this goal's action was executed, it moved the predator towards the prey. However, since the prey was moving faster it always returned a 'not achieved' result. Thus, the goal was never be deleted.

Hence, the goal and target were still in the goal queue as valid goals when the prey had moved beyond perception range. This allowed the predator to follow the prey even though it supposedly could not sense where it was.

One solution to this failing would be to modify the Motor Component so that it would not permit movements towards entities that were beyond perception range. However, this solution is not entirely satisfactory because the Behavioural Component would be unaware that its course of action was not allowed. It would thus continue to attempt to move towards the prey, preventing any lower-priority goals from being executed. If no goals of a higher priority appeared in the goal queue, the predator would appear to 'freeze' and would stop all movement. This issue could be resolved by returning the information from the Motor Component. However this solution would result in an implementation almost identical to

	Implementation 1	**Implementation 2**	**Implementation 3**
Predator / Prey	Predator chases prey. When prey successfully moves beyond perception range, the predator begins wandering randomly. Expected result.	Predator chases prey. Even when the prey has successfully moved beyond perception range, the predator is still able to pursue successfully. **Incorrect** result.	Predator chases prey. When prey successfully moves beyond perception range, the predator begins wandering randomly. Expected result.
Four entity fight	Entities move randomly. When they meet, they engage and one entity is deleted until there is only one entity left. Expected result.	Entities move randomly. When they meet, they engage and one entity is deleted until there is only one entity left. Expected result.	Entities move randomly. When they meet, they engage and one entity is deleted until there is only one entity left. Expected result.
Hungry / seeking food	Entity moves randomly until hunger level reaches a certain value. At this point, it seeks food and wanders again when food has been found. Expected result.	Entity moves randomly until hunger level reaches a certain value. At this point, it seeks food and wanders again when food has been found. Expected result.	Entity moves randomly until hunger level reaches a certain value. At this point, it seeks food and wanders again when food has been found. Expected result.
Zones of tolerance	Entity moves at random. When it strays too close, it moves away. When it strays too far, it moves towards. Expected result.	Entity moves at random until it strays too close, at which point it moves away indefinitely until it reaches the edge of the virtual world. **Incorrect** result.	Entity moves at random until it strays too close, at which point it moves away indefinitely until it reaches the edge of the virtual world. **Incorrect** result.

Results of testing animations under each of three implementations.
Table 1

implementation 3, except that the Components would exhibit greater coupling.

This would seem to indicate that only permitting a goal to be ‘achieved’ or ‘not achieved’, is not sufficient to create a usable generic Behavioural Component.

5.2 Limitations of not deleting until achieved or failed

In the zones of tolerance example, the animation did not proceed as expected under implementations 2 or 3. The expected behaviour was that the entity would move randomly. If it moved too close to the central creature, a ‘flee from’ goal would be triggered to move the entity away again. Similarly, if it moved too far away, a ‘go to’ goal would be triggered to move the entity closer. This expected behaviour was reflected under implementation 1.

However, under implementations 2 and 3 the entity moved randomly until the first time it strayed too close to the central creature. At this point, the entity moved away from the central creature and stopped only when it reached the edge of the virtual world and could move no further. It then remained at this point.

Under implementation 2, the cause of this was again due to there only being an ‘achieved’ or ‘not achieved’ state for each goal. When the entity strayed too close to the central creature, a ‘flee from’ goal was created. This was then repeatedly executed while the central creature was still within the perception range. Despite the fact that the entity had moved back into the zone of tolerance, and did not need to move further, the ‘flee from’ goal was not removed from the goal queue.

Once the entity had begun to move too far from the central creature, a ‘go to’ goal was also created. However, because this goal had a lower priority than ‘flee from’, it was never triggered. Simply making ‘go to’ the higher priority goal would not work either, because the entity would then display the opposite, and equally undesired, behaviour of moving towards the central object and remaining 'attached' to it. Giving them equal priorities and letting chance decide which is executed may be satisfactory for some purposes, but could not be depended upon to produce the required behaviour consistently.

A naïve solution to the problem would be to set the entity's perception range to the same distance as the inner distance of the zone of tolerance. This way, the goal would return 'achieved' as soon as it reached this distance and the 'flee from' goal would be deleted. However, this solution does not work because the entity is then unable to sense where the central object is and thus can not respond when it moves too far from the central object.

It would appear that in implementations 2 and 3, no arrangement of 'achieved', 'not achieved' or 'failed' results is sufficient to produce the range of results desired.

A better solution to the problem may be to produce a more flexible implementation of the goal deletion process, for example, by permitting the animator to enter a rule for each goal that would determine when the goal should be deleted. Such a method would allow goals to be deleted not only when they were achieved or failed, but when other conditions were met: in this example, it could be deleted when the entity was back within its zone of tolerance.

In this test case, if there had been a facility for deleting the goal when the entity was back within the zone of tolerance, implementation 3 and implementation 1 would have produced the same results. Thus it would appear that a more flexible rule-based process for removing goals would be capable of producing the same animations as the immediate deletion method of implementation 1.

The immediate deletion method of implementation 1, although its goal queue management is less efficient, produced the desired result in every case. This is because goals are deleted after one frame, meaning that the problems outlined above do not occur. Although implementation 1 could produce the same results as implementation 3, it is not certain that it could produce the full spectrum of results that could be achieved by enhancing implementation 3 with deletion rules as discussed above.

Thus, of the three implementations tested, implementation 1 offered the simplest and most consistent results by deleting goals after one cycle. However, as discussed above, if the goals are to remain in the goal for more than one cycle, it may be necessary to include a rule-based system in order to present the animator with the required genericity.

6. CONCLUSION

This paper investigated the hypothesis by Millar et al. [Milla99a] that it is possible to produce a generic Behavioural Animation system based on a four-component structure comprising Environment, Perception, Behavioural and Motor components. Millar et al. also presented a proposed structure for a generic Behavioural Component. Three different Behavioural Components, all following this structure but differing in their precise implementation details, were discussed and tested with the same animations.

It was shown that in some cases, the implementations produced different animations from the expected results. This was shown to be due to the manner in which goals are managed within each Behavioural Component, and in particular the circumstances under which they are deleted. The test animations demonstrated that the Behavioural Component displayed satisfactory genericity when goals were deleted after each cycle.

It can thus be concluded that the framework proposed by Millar et al. does provide a suitable structure for the production of a generic Behavioural Animation system.

REFERENCES

[Blumb95a] Blumberg,BM, Galyean,A: A multi-level direction of autonomous creatures for real-time virtual environments. *Proceedings of Computer Graphics*, pp47-54, 1995.

[Milla99a] Millar,RJ, Hanna,JRP, Kealy, SM: A review of Behavioural Animation, *Computers and Graphics*, Vol. 23, No. 1, pp127-143, 1999.

[Renau90a] Renault,O, Magnenat-Thalmann,N, Thalmann,D: A vision based approach to behavioural animation, *Journal of Visualization and Computer Animation*, Vol. 1, No. 1, pp 18-21, 1990.

[Reyno87a] Reynolds,CW*: Flocks, herds, schools: a distributed behavioural model, Proceedings of the SIGGRAPH '87 Computer Graphics*, Vol. 21, No. 4, pp25-34, 1987.

[Takeu92a] Takeuchi,T, Unuma,M, Amakawa,K: Path planning and its application to human animation systems, *Proceedings of Computer Animation '92*, pp163-175, 1992.

[Thoma84a] Thomas,F: Can classic Disney animation be duplicated on the computer?, *Computer Pictures*, Vol. 2, No. 4, pp20-26, 1984.