# Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments

Bruce M. Blumberg and Tinsley A. Galyean♣
MIT Media Lab

## ABSTRACT

There have been several recent efforts to build behavior-based autonomous creatures. While competent autonomous action is highly desirable, there is an important need to integrate autonomy with "directability". In this paper we discuss the problem of building autonomous animated creatures for interactive virtual environments which are also capable of being directed at multiple levels. We present an approach to control which allows an external entity to "direct" an autonomous creature at the motivational level, the task level, and the direct motor level. We also detail a layered architecture and a general behavioral model for perception and action-selection which incorporates explicit support for multi-level direction. These ideas have been implemented and used to develop several autonomous animated creatures.

## 1. INTRODUCTION

Since Reynold's seminal paper in 1987, there have been a number of impressive papers on the use of behavioral models to generate computer animation. The motivation behind this work is that as the complexity of the creature's interactions with its environment and other creatures increases, there is an need to "endow" the creatures with the ability to perform autonomous activity. Such creatures are, in effect, autonomous agents with their own perceptional, behavioral, and motor systems. Typically, authors have focused on behavioral models for a specific kind of creature in a given environment, and implemented a limited set of behaviors. There are examples of locomotion [2, 5, 7, 14, 16], flocking [18], grasping [9], and lifting [2]. Tu and Terzopoulos's Fish [20] represent one of the most impressive examples of this approach.

Advances in behavioral animation are critically important to the development of creatures for use in interactive virtual environments. Research in autonomous robots [4, 8, 12] supports the need to couple real-time action with dynamic and unpredictable environments. Their insights only serve to strengthen the argument for autonomous animated creatures.

Pure autonomy, perhaps, should not be the ultimate goal. Imagine making an interactive virtual "Lassie" experience for children. Suppose the autonomous animated character playing Lassie did a fine job as a autonomous dog, but for whatever reason was ignoring the child. Or suppose, you wanted the child to focus on some aspect of the environment which was important to the story, but Lassie was distracting her. In both cases, you would want to be able to provide external control, in real-time, to the autonomous Lassie. For example, by increasing its "motivation to play", it would be more likely to engage in play. Alternatively, Lassie might be told to "go over to that tree and lie down" so as to be less distracting.

Thus, there is a need to understand how to build animated characters for interactive virtual environments which are not only capable of competent autonomous action but also capable of responding to external control. We call this quality "directability." This is the fundamental problem addressed in this paper.

This paper makes 3 primary contributions to the body of literature regarding animated autonomous characters. Specifically, we describe:

- An approach to control which allows an external entity to "direct" a virtual character at a number of different levels.
- A general behavioral model for perception and action-selection in autonomous animated creatures but which also supports external control.
- A layered architecture which supports extensibility, reusability and multiple levels of direction.

An experimental toolkit which incorporates these ideas has been successfully used to build a number of creatures: a virtual dog used in an interactive virtual environment, and several creatures used in an interactive story telling environment.

The remainder of the paper is organized as follows. In section 2 we present a more detailed problem statement and summarize the key contributions of our approach. In section 3 we present an overview of the general architecture. In sections 4, 5 and 6 we discuss the motor, sensory and behavior systems in more depth. In section 7 we discuss how directability is integrated into our architecture. Finally, in section 8 we discuss some aspects of our implementation and give examples of its use.

## 2. PROBLEM STATEMENT

An autonomous agent is a software system with a set of goals which it tries to satisfy in a complex and dynamic environment. It is autonomous in the sense that it has mechanisms for sensing and interacting with its environment, and for deciding what actions to take so as to best achieve its goals[12]. In the case of an autonomous animated creature, these mechanisms correspond to a set of sensors, a motor system and associated geometry, and lastly a behavior system. In our terminology, a creature is an animate object capable of goal-directed and time-varying behavior.

Deciding on the "right" action or set of actions is complicated by a number of factors. For example, due to the problems inherent in sensing and perception, a creature's perception of its world is likely to be incomplete at best, and completely erroneous at worst.

♣MIT Media Lab, 20 Ames St., Cambridge MA, 02139.
bruce/tag@media.mit.edu

Published in Proceedings of SIGGRAPH 95, ACM *Computer Graphics*, August, 1995.

There may be competing goals which work at cross-purposes (e.g. moving toward food may move the creature away from water). This can lead to dithering in which the creature oscillates among competing activities. On the other hand, an important goal may be un-obtainable, and pursuit of that goal may prevent the satisfaction of lower priority, but attainable goals. External opportunities need to be weighed against internal needs in order to provide just the right level of opportunistic behavior. Actions may be unavailable or unreliable. To successfully produce competent autonomous action over extended periods of time, the Behavior System must provide solutions to these problems, as well as others.

However, as mentioned earlier, strict autonomy is not the goal. We need, in addition, to direct the creature at a number of different levels. Three levels of input, (motivational, task, and direct) are outlined in Figure 1. Additionally, commands at the direct level need to be able to take three imperative forms:

- Do it, independent of the Behavior System.
- Do it, if the Behavior System doesn't object.
- Suggest how an action should be performed, should the Behavior System wish to perform that action.

Thus, the behavior and motor systems must be designed and implemented in such a way that it is possible to support these levels and types of direction at run-time.

Building autonomous animated creatures is inherently an iterative process. This is particularly true since we are in the early phases of understanding how to build them. Ideally, a common approach should be taken for the specification of geometry through to behavior so that a developer need only learn a single framework. Lastly, an embedded interpreter is required to facilitate testing, as well as run-time direction.

### 2.1 Multiple Levels of Control

We provide an approach to control which allows an external entity to "direct" an autonomous animated creature at a number of different levels. These levels are detailed in Figure 1. By providing

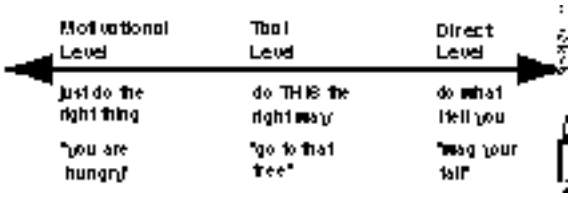


**Figure 1**: Here we articulate three levels at which a creature can be directed. At the highest level the creature would be influenced by changing its current motivation and relying on it to react to this change. If you tell it to be hungry it will go off looking for food. At the task level you give it a high level directive and you expect it to carry out this command in a reasonable manner (for example walking around a building instead of through it.) At the lowest level you want to give a creature a

the ability to "direct" the creature at multiple levels the animator or developer can choose the appropriate level of control for a given situation. Both Badler and Zeltzer have proposed similar decomposition of control [2, 23].

### 2.2 A General Behavior Model

We propose a distributed behavioral model, inspired by work in Ethology and autonomous robot research, for perception and action-selection in autonomous animated creatures but which also supports external control. The contributions of this model include:

- A general model of action-selection which provides greater control over temporal patterns of behavior than previously described approaches have offered.
- A natural and general way to model the effect of external stimuli and internal motivation.
- An approach in which multiple behaviors may suggest actions to be performed and preferences for how the actions are to be executed, while still maintaining the advantages of a winner-take-all architecture.
- An implementation which supports motivational and task level direction at run-time.

We also describe a robotics inspired approach to low-level autonomous navigation in which creatures rely on a form of synthetic vision to perform navigation and obstacle avoidance.

### 2.3 A Layered Architecture

A 5-layered architecture for autonomous animated creatures is described. Several important abstraction barriers are provided by the architecture:

- One between the Behavior System and the Motor Skills, which allows certain behaviors (e.g. "move-toward") to be independent of the Motor Skills which perform the desired action (e.g. "drive" vs. "walk") in a given creature.
- One between the Motor Skills and geometry which serves as both an abstraction barrier and a resource manager.

The result is an architecture which encourages re-usability and extensibility, while providing the necessary foundation to support autonomous action with interactive direction.

## 3. ARCHITECTURE

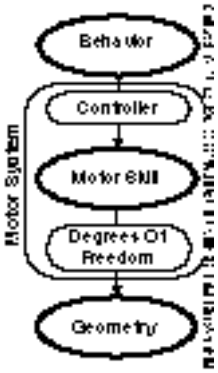


**Figure 2:** Block diagram of a creature's architecture. The basic structure consists of the three basic parts (Geometry, Motor Skills and Behavior) with two layers of abstraction between these parts (Controller, and Degrees of Freedom).

Figure 2 shows the basic architecture for a creature.The geometry provides the shapes and transforms that are manipulated over time for animation. The Motor Skills provide atomic motion elements which manipulate the geometry in order to produce coordinated motion. "Walking" or "Wagging the tail" are examples of Motor Skills. Motor Skills manipulate the geometry with no knowledge of the environment or state of a creature, other than that needed to execute the skill. At the top rests the Behavior System of a creature. This element is responsible for deciding what to do, given its goals and sensory input and triggering the correct Motor Skills to achieve the current task or goal. In addition to these three parts, there are two layers of insulation, the controller and the degrees of freedom (DOFs), which are important to making this architecture generalizable and extensible.

Behaviors implement high level capabilities such as, "find food and eat", or "sit down and shake", as well as low level capabilities such as "move to" or "avoid obstacle" by issuing the appropriate motor commands (i.e "forward", "left", "sit", etc.) to the controller. Some behaviors may be implemented in a creature-independent way. For example, the same "move to" behavior may be applicable to any creature with basic locomotive skills (e.g. forward, left, right,...) although each may use different Motor Skills to perform the required action. It is the controller which provides this common interface to the Motor Skills by mapping a generic command ("forward") into the correct motor skill(s) and parameters for

a given creature. In this way, the same behavior may be used by more than one type of creature.

Figure 3 shows the sources of input to the creature. Sensors are

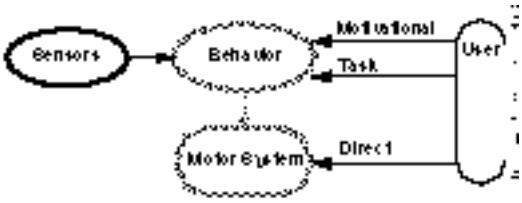


**Figure 3**: There are two sources of input to a creature. First are sensors associated with the creature. These sensors are used by the Behavior System to enable both task level and autonomous behavior. The other source of input is from the user (or application using the creature.) This input can happen at multiple levels, ranging from simply adjusting a creature's current motivational state to directly turning a motor skill on or off.

elements of a creature which the creature uses to interrogate the environment for relevant information. The creature may also take additional input from the user or the application using the creature. These directives can enter the creature's computational model at the three different levels.

## 4. MOTOR SYSTEM

We use the term "motor system" to refer to the three layers that lie between the Behavior System and the geometry, Figure 2. These parts include the Motor Skills in the center, and the abstraction and interface barriers on either side of the Motor Skills. Together these three layers of the architecture provide the mapping from motor commands to changes in the geometry over time.

The motor system is designed to meet the following 5 important criteria:

- Act as an abstraction barrier between high-level commands (e.g. "forward") and the creature specific implementation (e.g. "walking").
- Support multiple imperative forms for commands.
- Provide a generic set of commands which all creatures can perform.
- Minimize the amount of "house-keeping" required of the Behavior System.
- Provides resource management so as to support coherent, concurrent motion.

Within these three layers of the motor system, the controller provides the high level abstraction barrier and the support of multiple imperative forms. Motor skills can be inherited allowing basic skills to be shared amongst creatures. Also, Motor Skills are designed to minimize the "house-keeping" that an external user or Behavior System must do. It is the degree of freedom abstraction barrier that serves as the resource manager.

### 4.1 Degrees of Freedom (DOFs)

Degrees of Freedom (DOFs) are "knobs" that can be used to modify the underlying geometry. They are the mechanism by which creatures are repositioned and reshaped. For example, DOFs might be used to wag the tail, move a joint, or reposition an entire leg. DOFs serve 2 important functions:

- Resource management. Provides a locking mechanism so that competing Motor Skills do not conflict.
- An abstraction barrier. Utilizes interpolators to re-map simple input values (0 to 1) to more complex motion.

The resource management system is a simple one. Each DOF can be locked by a motor skill, restricting it by anyone else until unlocked. This locking provides a mechanism for insuring coherent, concurrent motion. As long as two or more Motor Skills do not conflict for DOFs they are free to run concurrently. Alternatively, if a motor skill requests DOFs that are already locked it will be informed it cannot run currently.

When functioning as an abstraction barrier a DOF provides a mechanism to map a simple input value (often a number between 0 and 1) to another space via interpolators and inverse kinematics such as in, Figure 4. It is this abstraction that allows a motor skill

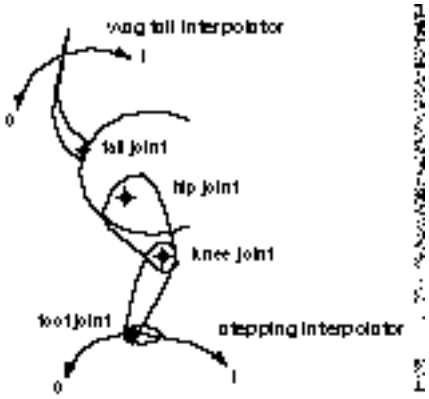


**Figure 4**: DOFs in a creature can provide interfaces to the geometry at several different levels. For example, joints (and therefore the associated transformations) can be directly controlled or indirectly as is the case with this leg. Here inverse kinematics is used to move the foot. An additional level of abstraction can be added by using interpolators. The interpolator on the leg provides a "one knob" interface for the motor skill. By giving a number between 0 and 1 the motor skill can set the location of the leg along one stepping cycle. Likewise the tail can be wagged with only one number.

to position a leg along a stepping cycle via one number. Note that when a high level DOF (the stepping DOF in this example) is locked the lower level DOFs it utilizes (the leg joints) are in turn locked.

### 4.2 Motor Skills

A motor skill utilizes one or more DOFs to produce coordinated movement. Walking, turning, or lower head are all examples of Motor Skills. A motor skill can produce complicated motion, and the DOFs' locking mechanism insures that competing Motor Skills are not active at the same time. In addition, Motor Skills present an extremely simple interface to upper layers of the architecture. A motor skill can be requested to turned on, or to turn off. In either case, arguments may be passed as part of the request.

Motor skills rely heavily on degrees of freedom to do their work. Each motor skill declares which DOFs it needs in order to perform its task. It can only become active if all of these DOFs are unlocked. Once active, a motor skill adjusts all the necessary DOFs with each time-step to produce coordinated animation.

Most Motor Skills are "spring-loaded." This means that if they have not been requested to turn on during an update cycle, they begin to move their DOFs back toward some neutral position and turn off within a few time-steps.The advantage of this approach is that a behavior, which turns on a skill, need not be concerned with turning it off at the correct time. The skill will turn itself off, thereby reducing the amount of "bookkeeping." Because Motor Skills are "spring-loaded" the Behavior System is required to specify which skills are to be active with each time-step. This may seem like a burden but it is consistent with a reactive behavior system which re-evaluates what actions it should perform during every update cycle. It should also be noted that this spring-loaded feature can be turned off, to facilitate sources of direction other than the Behavior System.

There are a number of basic Motor Skills which all creatures inherit, such as ones for setting the position or heading of the creature.

### 4.3 Controller

The controller is a simple but significant layer in the architecture which serves an important function as an abstraction barrier

between the Behavior System and the underlying Motor Skills. The primary job of the controller is to map commands such as "forward", "turn", "halt", "look at" etc. into calls to turn on or turn off the appropriate motor skill(s). Thus, "forward" may result in the "walk" motor skill being turned on in the dog but the "move" motor skill in the case of the car. This is an important function because it allows the Behavior System or application to use one set of commands across a potentially wide-range of creatures, and lets the motor system of each creature to interpret them differently but appropriately.

The controller accepts commands in the form of a data structure called a motor command block. This data structure specifies the command and any arguments. In addition, a motor command block can store return arguments, allowing functions that inquire about the state of a creature (its position, its velocity) to be treated by the same mechanism as all other commands. A command block can be issued to the controller as one of three imperative forms: primary command - to be executed immediately; secondary - to be queued at a lower priority; and as a meta command - suggesting how another command should be run, Figure 5. These different imperative forms are used extensively by the Behavior System (see section 6.6). They allow multiple behaviors to simultaneously express their preferences for motor actions.

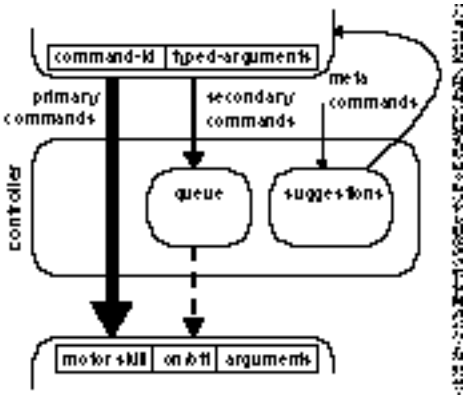


**Figure 5**: An incoming command is represented in a motor command block consisting of a command id and an optional list of typed arguments. If these arguments are not supplied then defaults stored in the controller are used. Any given command will turn on or off one or more Motor Skills, while also providing any necessary arguments. Commands take two levels of importance. Primary commands are executed right away, while secondary commands are queued. These queued commands are executed at the end of each time cycle, and only if the necessary resources (DOFS) are available. It is expected that these secondary commands will be used for suggested but not imperative actions. The last type of input into the controller is in the form of a meta-command. These commands are stored as suggestion of how to execute a command. For example, "if you are going to walk I suggest that you walk slowly." These are only stored in the controller and it is the responsibility of the calling application (or user) to use or ignore a suggestion.

## 5. SENSING

There are at least three types of sensing available to autonomous animated creatures:

- Real-world sensing using real-world "noisy" sensors.
- "Direct" sensing via direct interrogation of other virtual creatures and objects.
- "Synthetic Vision" in which the creature utilizes vision techniques to extract useful information from an image rendered from their viewpoint.

While it is important to support all three types of sensing, we have found synthetic vision to be particularly useful for low-level navigation and obstacle avoidance. Several researchers, including Renault [17], Reynolds[18], and Latombe[10] have suggested similar approaches.

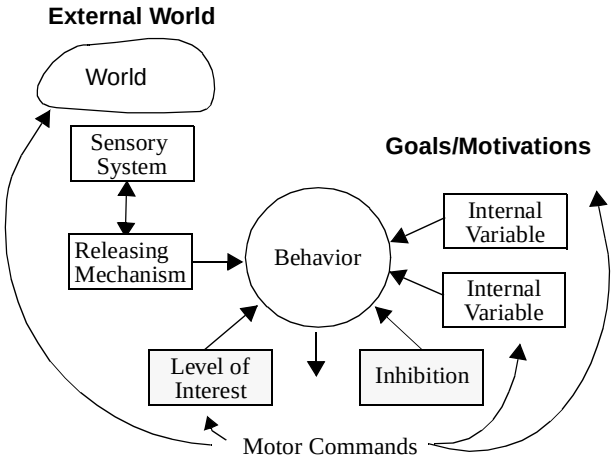


**Figure 6**:The purpose of a Behavior is to evaluate the appropriateness of the behavior, given external stimulus and internal motivations, and if appropriate issue motor commands. Releasing Mechanisms act as filters or detectors which identify significant objects or events from sensory input, and which output a value which corresponds to the strength of the sensory input. Motivations or goals are represented via Internal Variables which output values which represents the strength of the motivation. A Behavior combines the values of the Releasing Mechanisms and Internal Variables on which it depends and that represents the value of the Behavior before Level of Interest and Inhibition from other Behaviors. Level of Interest is used to model boredom or behavior-specific fatigue. Behaviors must compete with other behaviors for control of the creature, and do so using Inhibition (see text for details).There are a variety of explicit and implicit feedback mechanisms.

### 5.1 Synthetic Vision For Navigation

Horswill [8] points out that while "vision" in general is a very hard problem, there are many tasks for which it is possible to use what he calls "light-weight" vision. That is, by factoring in the characteristics of the robot's interaction with the environment and by tailoring the vision task to the specific requirements of a given behavioral task, one can often simplify the problem. As a result, vision techniques developed for autonomous robots tend to be computationally cheap, easy to implement, and reasonably robust.

Synthetic vision makes sense for a number of reasons. First, it may be the simplest and fastest way to extract useful information from the environment (e.g. using vision for low-level obstacle avoidance and navigation versus a purely analytical solution). This may be particularly true if one can take advantage of the rendering hardware. Moreover, synthetic vision techniques will probably scale better than analytical techniques in complex environments. Finally, this approach makes the creature less dependent on the implementation of its environment because it does not rely on other creatures and objects to respond to particular queries.

Our approach is simple. The scene is rendered from the creature's eye view and the resulting image is used to generate a potential field from the creature's perspective (this is done in an approach similar to that of Horswill). Subsequently, a gradient field is calculated, and this is used to derive a bearing away from areas of high potential. Following Arkin [1], some behaviors within the Behavior System represent their pattern of activity as a potential fields as well (for example, moveto). These potential fields are combined with the field generated by the vision sensor to arrive at a compromise trajectory.

This sensor is a simple example of using a technique borrowed from robotics. It was simple to implement, works well in practice, and is general enough to allow our virtual dog to wander around in new environments without modification.

## 6. BEHAVIOR SYSTEM

The purpose of the Behavior System is to send the "right" set of control signals to the motor system at every time-step. That is, it must weigh the potentially competing goals of the creature, assess

the state of its environment, and choose the set of actions which make the "most sense" at that instant in time. More generally, it provides the creature with a set of high-level behaviors of which it is capable of performing autonomously in a potentially unpredictable environment. Indeed, it is this ability to perform competently in the absence of external control which makes high level motivational or behavioral control possible.

Action-selection has been a topic of some interest among Ethologists and Computer Scientists alike, and a number of algorithms have been proposed [3, 4, 12, 19-22]. Earlier work [3], presented a computational model of action-selection which draws heavily on ideas from Ethology. The algorithm presented below is derived from this work but incorporates a number of important new features. The interested reader may consult [3] for the ethological justification for the algorithm. The remainder of this section describes the major components of the Behavior System, and how it decides to "do the right thing"

### 6.1 Behaviors

While we have spoken of a Behavior System as a monolithic entity, it is in fact a distributed system composed of a loosely hierarchical network of "self-interested, goal-directed entities" called Behaviors. The granularity of a Behavior's goal may vary from very general (e.g. "reduce hunger") to very specific (e.g. "chew food"). The major components of an individual Behavior are shown in Figure 6. This model of a distributed collection of goal-directed entities is consistent with ethological models as well as recent theories of the mind [15].

Behaviors compete for control of the creature on the basis of a value which is re-calculated on every update cycle for each Behavior. The value of a Behavior may be high because the Behavior satisfies an important need of the creature (e.g. its Internal Variables have a high value). Or it may be high because the Behavior's goal is easily achievable given the Behavior's perception of its environment (e.g. its Releasing Mechanisms have a high value).

Behaviors influence the system in several ways: by issuing motor commands which change the creature's relationship to its environment, by modifying the value of Internal Variables, by inhibiting other Behaviors, or by issuing suggestions which influence the motor commands issued by other Behaviors.

Behaviors are distinguished from Motor Skills in two ways. First, a Behavior is goal-directed whereas a Motor Skill is not. For example, "Walking" in our model is a Motor Skill. "Moving toward an object of interest" is a Behavior. Second, a Behavior decides when it should become active, whereas a Motor Skill runs when something else decides it should be active. Typically, Behaviors rely on Motor Skills to perform the actions necessary to accomplish the Behavior's goals.

### 6.2 Releasing Mechanisms and Pronomes

Behaviors rely on objects called "Releasing Mechanisms" to filter sensory input and identify objects and/or events which are relevant to the Behavior, either because they are important to achieving the Behavior's goal, or because their presence determines the salience of the Behavior given the creature's immediate environment. Releasing Mechanisms output a continuous value which typically depends on whether the stimuli was found, on its distance and perhaps on some measure of its quality. This is important because by representing the output of a Releasing Mechanism as a continuous quantity, the output may be easily combined with the strength of internal motivations which are also represented as continuous values. This in turn allows the creature to display the kind of behavior one finds in nature where a weak stimulus (e.g. day-old pizza) but a strong motivation (e.g. very hungry) may result in the same behavior as a strong stimulus (e.g. chocolate cake) but weak motivation (e.g. full stomach).

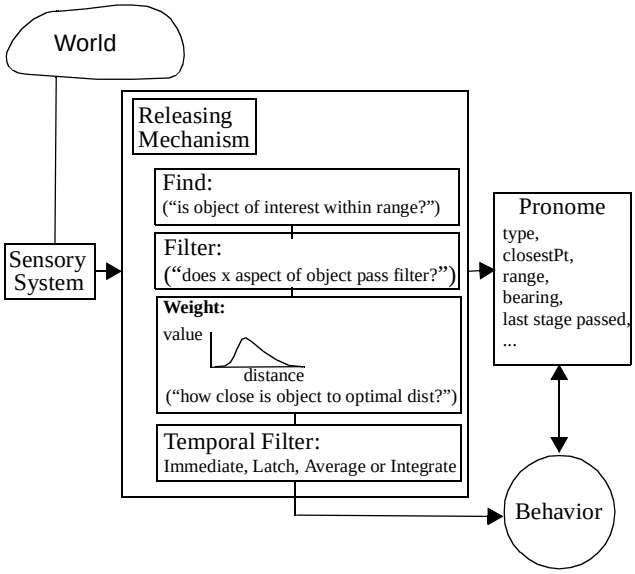


**Figure 7**: Releasing Mechanisms identify significant objects or events from sensory input and output a value which represents the strength of the stimulus. By varying the allowed maximum for a given Releasing Mechanism, a Behavior can be made more or less sensitive to the presence of whatever input causes the Releasing Mechanism to have a non-zero value. A Releasing Mechanism has 4 phases (Find, Filter, Weight and Temporal Filtering), as indicated above, each of which is implemented by callbacks. Releasing Mechanisms can often share the same generic callback for a given phase. Temporal Filtering is provided to deal with potentially noisy data.

While Releasing Mechanisms may be looking for very different objects and events, they typically have a common structure. This is described in more detail in Figure 7. The importance of this is that it is possible to share functionality across Releasing Mechanisms.

In addition to transducing a value from sensory input, a Releasing Mechanism also fills in a data structure available to the Behavior called a Pronome [15]. The Pronome acts like a pronoun in English: The use of Pronomes makes it possible for the Behavior to be written in terms of "it", where "it" is defined by the Behavior's Pronome. Thus, a "stopNearAndDo" Behavior can be implemented without reference to the kind of object it is stopping near. Pronomes may be shared among behaviors, thus allowing the construction of a generic "find and do" hierarchy. While the motivation for Pronomes comes from theories of mind [15] as opposed to ethology, they make sense in an ethological context as well. In any event, the use of Pronomes greatly facilitates the integration of external control, and simplifies the construction of behavior networks by providing a level of abstraction.

### 6.3 Internal Variables

Internal Variables are used to model internal state. Like Releasing Mechanisms, Internal Variables express their value as a continuous value. This value can change over time based on autonomous growth and damping rates. In addition, Behaviors can potentially modify the value of an Internal Variable as a result of their activity.

Both Releasing Mechanisms and Internal Variables may be shared by multiple Behaviors.

### 6.4 Behavior Groups

Behaviors are organized into groups of mutually inhibiting behaviors called Behavior Groups as shown in Figure 8. While we find a loose hierarchical structure useful this is not a requirement (i.e. all the Behaviors can be in a single Behavior Group). Behavior Groups are important because they localize the interaction among Behaviors which facilitates adding new Behaviors.

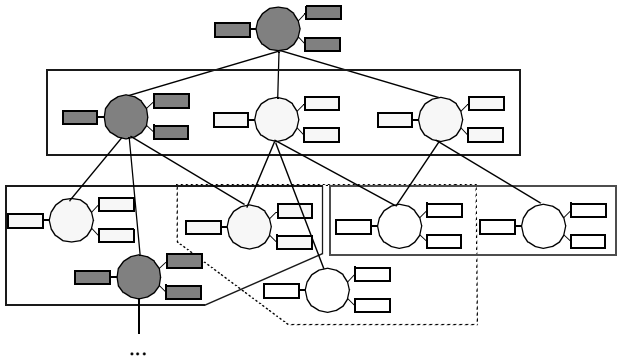

**Figure 8**:Behaviors are organized into groups of mutually inhibiting Behaviors called Behavior Groups. These Behavior Groups are in turn organized in a loose hierarchical fashion. Behavior Groups at the upper levels of the hierarchy contain general types of behaviors (e.g. "engage-in-feeding") which are largely driven by motivational considerations, whereas lower levels contain more specific behaviors (e.g. "pounce" or "chew") which are driven more by immediate sensory input. The arbitration mechanism built into the algorithm insures that only one Behavior in a given Behavior Group will have a non-zero value after inhibition. This Behavior is then active, and may either issue primary motor commands, or activate the Behavior Group which contains its children Behaviors (e.g. "search-for-food", "sniff", "chew" might be the children of "engage-in-feeding"). The dark gray behaviors represent the path of active Behaviors on a given tick. Behaviors which lose to the primary Behavior in a given Behavior Group may nonetheless influence the resulting actions of the creature by issuing either secondary or meta-commands.

### 6.5 Inhibition and Level of Interest

A creature has only limited resources to apply to satisfying its needs (e.g. it can only walk in one direction at a time), and thus there needs to be some mechanism to arbitrate among the competing Behaviors. Moreover, once a creature is committed to satisfying a goal, it makes sense for it to continue pursuing that goal unless something significantly more important comes along.

We rely on a phenomena known as the "avalanche effect" [15] to both arbitrate among Behaviors in a Behavior Group and to provide the right amount of persistence. This is done via mutual inhibition. Specifically, a given Behavior A will inhibit a Behavior B by a gain $I_{AB}$ times Behavior A's value. By (a) restricting the inhibitory gains to be greater than 1, (b) by clamping the value of a Behavior to be 0 or greater, and (c) requiring that all Behaviors inhibit each other, the "avalanche effect" insures that once the system has settled, only one Behavior in a Behavior Group will have a non-zero value. This model of inhibition was first proposed, in an ethological context by Ludlow [11], but see Minsky as well.

This model provides a robust mechanism for winner-take-all arbitration. It also provides a way of controlling the relative persistence of Behaviors via the use of inhibitory gains. When the gains are low, the system will tend to dither among different behaviors. When the gains are high, the system will show more persistence.

The use of high inhibitory gains can, however, result in pathological behavior in which the creature pursues a single, but unattainable goal, to the detriment of less important, but achievable ones. Ludlow addressed this problem by suggesting that a level of interest be associated with every Behavior. It is allowed to vary between 0 and 1 and it has a multiplicative effect on the Behavior's value.When Behavior is active the level of interest decreases which in turn reduces the value of the Behavior regardless of its intrinsic value. Eventually, this will allow another Behavior to become active (this is known as time-sharing in the ethological literature). When the behavior is no longer active, its level of interest rises.

Inhibitory gains and level of interest provide the designer with a good deal of control over the temporal aspects of behavior. This is an important contribution of this algorithm.

### 6.6 Use of Primary, Secondary and Meta-commands

Being active means that a Behavior or one if its children has top priority to issue motor commands. However, it is extremely important that less important Behaviors (i.e. those which have lost the competition for control) still be able to express their preferences for actions. This is done by allowing Behaviors which lose to issue suggestions in the form of secondary and meta-commands as described earlier. These suggestions are posted prior to the winning Behavior taking its action, so it can utilize the suggestions as it sees fit.

For example, a dog may have a behavior which alters the dog's characteristics so as to reflect its emotional state. Thus, the behavior may issue secondary commands for posture as well as for the desired ear, tail and mouth position, and use a meta-command for the desired gait. The use of a meta-command for gait reflects the fact that the behavior may not know whether the dog should go forward or not. However, it does know how it wants the dog to move, should another behavior decide that moving makes sense.

Despite the use of secondary and meta-commands, the winning behavior still has ultimate say over what actions get performed while it is active. It can over-rule a secondary command by removing it from the queue or by executing a Motor Skill which grabs a DOF needed by a given secondary command. In the case of meta-commands, the winning behavior can choose to ignore the meta-command, in which case it has no effect.

### 6.7 The Algorithm

The action-selection algorithm is described below. The actual equations are provided in appendix A.

On each update cycle:

(1) All Internal Variables update their value based on their previous value, growth and damping rates, and any feedback effects.

(2) Starting at the top-level Behavior Group, the Behaviors within it compete to become active. This is done as follows:

(3) The Releasing Mechanisms of each Behavior update their value based on the current sensory input. This value is then summed with that of the Behavior's Internal Variables and the result is multiplied by its Level Of Interest. This is repeated for all Behaviors in the group.

(4) For each Behavior in the group, the inhibition due to other Behaviors in the group is calculated and subtracted from the Behavior's pre-inhibition value and clamped to 0 or greater.

(5) If after step (4) more than one Behavior has a non-zero value then step (4) is repeated (using the post-inhibition values as the basis) until this condition is met. The Behavior with a non-zero value is the active Behavior for the group.

(6) All Behaviors in the group which are not active are given a chance to issue secondary or meta-commands. This is done by executing a suggestion callback associated with the Behavior.

(7) If the active Behavior has a Behavior Group as a child (i.e. it is not a Behavior at the leaf of the tree), then that Behavior Group is made the current Behavior Group and the process is repeated starting at step (3). Otherwise, the Behavior is given a chance to issue primary motor commands via the execution of a callback associated with the Behavior.

## 7. INTEGRATION OF DIRECTABILITY

Having described the Behavior and Motor Systems we are now in a position to describe how external control is integrated into this architecture. This is done in a number of ways.

First, motivational control is provided via named access to the Internal Variables which represent the motivations or goals of the Behavior System. By adjusting the value of a given motivational variable, the creature can be made more or less likely to engage in Behaviors which depend on that variable.

Second, all the constituent parts of a Behavior are also accessible at run-time, and this provides another mechanism for exerting

behavioral control. For example, by changing the type of object for which the Releasing Mechanism is looking, the target of a given Behavior can easily be altered (e.g. “fire hydrants” versus “user’s pants leg”). In addition, a Behavior may be made more or less opportunistic by adjusting the maximum allowed “strength” of its Releasing Mechanisms. A Behavior can be made in-active by setting its level of interest to zero.

Third, the Behavior System is structured so that action-selection can be initiated at any node in the system. This allows an external entity to force execution of a particular part or branch of the Behavior System, regardless of motivational and sensory factors which might otherwise favor execution of other parts of it. Since branches often correspond to task-level collections of Behaviors, this provides a form of task-level control.

Forth, it is easy to provide “imaginary” sensory input which in turn may trigger certain behaviors on the part of the creature. For example, objects may be added to the world which are only visible to a specific creature.The advantage of this technique is that it does not require the external entity to know anything about the internal structure of the creature’s Behavior System.

The mechanisms described above for controlling the Behavior System naturally support both prescriptive and proscriptive control. For example, by adjusting the level of a motivational variable which drives a given branch of the Behavior System, the director is expressing a weighted preference for or against the execution of that behavior or group of behaviors.

The multiple imperative forms supported by the Motor Controller allow the director to express weighted preferences directly at the motor level. For example, at one extreme, the director may “shut off” the Behavior System and issue motor commands directly to the creature. Alternatively, the Behavior System can be running, and the director may issue persistent secondary or meta-commands which have the effect of modifying or augmenting the output of the Behavior System. For example, the external entity might issue a secondary command to “wag tail”. Unless this was explicitly over-ruled by a Behavior in the Behavior System, this would result in the Dog wagging its tail. External meta-commands may also take the form of spatial potential field maps which can be combined with potential field maps generated from sensory data to effectively attract or repel the creature from parts of its environment.

## 8. IMPLEMENTATION

These ideas have been implemented as part of an object-oriented architecture and toolkit for building and controlling autonomous animated creatures. This toolkit is based on Open Inventor 2.0. Most of the components from which one builds a creature, including all of the components of the Behavior System, and the action-selection algorithm itself are derived from Inventor classes.This allows us to define most, of a creature and its Behavior System via a text file using the Inventor file format. This is important for rapid prototyping and quick-turnaround, as well as to facilitate the use of models generated via industry-standard modelers. We also make extensive use of Inventor’s ability to provide named access to field variables at run-time. This is important for integration of external control. Callbacks are used to implement most of the custom functionality associated with specific Releasing Mechanisms and Behaviors. This coupled with extensive parameterization reduces the need to create new subclasses. Lastly, an embedded Tcl interpreter is provided for interactive run-time control.

We have developed several creatures using this tool kit. For the Alive project [13], we have developed “Silas T. Dog” an autonomous animated dog which interacts with a user in a 3D virtual world in a believable manner. Silas responds to a dozen or so gestures and postures of the user, and responds appropriately (e.g. if the user bends over and holds out her hand, Silas moves toward the outstretched hand and eventually sits and shakes his paw). The dog always looks at its current object of interest (head, hand, etc.), and when it is sad or happy, its tail, ears and head move appropriately. Silas has a number of internal motivations which he is constantly trying to satisfy. For example, if his desire to fetch is high, and the user has not played ball with him, he will find a ball and drop it at the person’s feet. Similarly, he will periodically take a break to get a drink of water, or satisfy other biological functions.

Silas represents roughly 3000 lines of C++ code, of which, 2000 lines are for implementing his 24 dog specific Motor Skills. He responds to 70 motor commands. His Behavior System is comprised of roughly 40 Behaviors, 11 Behavior Groups, 40 Releasing Mechanisms and 8 Internal Variables. Silas runs at 15Hz on a Onyx Reality Engine with rendering and sensing time (i.e. the second render for his synthetic vision) comprising most of the update time. The evaluation of the Behavior System itself typically takes less than 6-8 milliseconds.

We have also developed a number of creatures which are used in the context of an interactive story system [6]. This system features a computational director which provides “direction” to the creatures so as to meet the requirements of the story. For example, at the beginning of the story, a dog hops out of a car and wanders around. If the user, who is wearing a head-mounted display, does not pay attention to the dog, the director will send the dog over to the user. If the user still does not pay attention, the director effectively tells the dog: “the user’s leg is a fine replacement for a hydrant, and you really have to...”. The resulting behavior on the part of the dog usually captures the user’s attention.

**Figure 9:** Silas and his half brother Lucky

## 9. CONCLUSION

Autonomy and directability are not mutually exclusive. We have detailed an architecture and a general behavioral model for perception and action-selection which can function autonomously while accepting direction at multiple levels. This multi-level direction allows a user to direct at whatever level of detail is desirable. In addition, this blend of autonomy and direction is demonstrated with several creatures within the context of two applications.


### Acknowledgments

The authors would like to thank Professors Pattie Maes, Alex P. Pentland and Glorianna Davenport for their support of our work. Thanks go, as well, to the entire ALIVE team for their help. In particular, thanks to Bradley Rhodes for suggesting the use of pronomes. We would also like to thank Taylor Galyean and Marilyn Feldmeier for their work modeling the setting and “Lucky.” This work was funded in part by Sega North America and the Television of Tomorrow Consortium.

## Appendix A.

**Behavior Update Equation:**

$$v_{it} = Max\left[\left(li_{it} \cdot Combine(\sum_k rm_{ki}, \sum_j iv_{jt}) - \sum_m n_{mi} \cdot v_{mt}\right), 0\right]$$

Where at time *t* for Behavior *i*,$v_{it}$ is its value; $li_{it}$ is the level of interest; $rm_{kt}$ and $iv_{jt}$ are the values of Releasing Mechanism *k*, and Internal Variable *j*, where *k* and *j* range over the Releasing Mechanisms and Internal Variables relevant to Behavior *i*; $n_{mi}$ (n>1) is the Inhibitory Gain that Behavior *m* applies against Behavior *i*; $v_{mt}$ is the value of Behavior *m*, where *m* ranges over the other Behaviors in the current Behavior Group. *Combine()* is the function used to combine the values of the Releasing Mechanisms and Internal Variables for Behavior i (i.e addition or multiplication).

**Internal Variable Update Equations:**

$$iv_{it} = (iv_{i(t-1)} \cdot damp_i) + growth_i - \sum_k effects_{kit}$$

Where at time *t* for Internal Variable *i*,$iv_{it}$ is its value; $iv_{i(t-1)}$ is its value on the previous time step; $damp_i$ and $growth_i$ are damping rates and growth rates associated with Internal Variable *i*; and $effects_{kit}$ are the adjustments to its value due to the activity of Behavior *k*, where *k* ranges over the Behaviors which directly effect its value when active.

$$effects_{kit} = (modifyGain_{ki} \cdot v_{k(t-1)})$$

Where $effects_{kit}$ is the effect of Behavior *k* on Internal Variable *i* at time *t*; $modifyGain_{ki}$ is the gain used by Behavior *k* against Internal Variable *i* and $v_{k(t-1)}$ is the value of Behavior *k* in the preceding time step.

**Level of Interest Update Equation:**

$$li_{it} = Clamp(((li_{i(t-1)} \cdot damp_i) + growth_i - (v_{i(t-1)} \cdot bRate_i)),0, 1)$$

Where $li_{it}$ is the Level Of Interest of Behavior *i* at time *t*, and $bRate_i$ is the boredom rate for Behavior *i*. Clamp(x,y,z) clamps x to be between y and z. Note Level Of Interest is just a special case of an Internal Variable.

**Releasing Mechanism Update Equation:**

$$rm_{it} = Clamp(TemporalFilter(t, rm_{i(t-1)}, Find(s_{it}, dMin_i, dMax_i) \cdot Filter(s_{it}) \cdot Weight(s_{it}, dOpt_i)), min_i, max_i)$$

Where $rm_{it}$ is the value of Releasing Mechanism *i* at time *t*; $s_{it}$ is the relevant sensory input for i; $dMin_i$ and $dMax_i$ are minimum and maximum distances associated with it; *Find()* returns 1 or 0 if the object of interest is found within $s_{it}$ and within $dMin_i$ to $dMax_i$; *Filter()* returns 1 or 0 if the object of interest matches some additional criteria; *Weight()* weights the strength of the stimulus based on some metric such as optimal distance $dOpt_i$; *TemporalFilter()* applies a filtering function (latch, average, integration, or immediate) over some period *t*; and *Clamp()* clamps the resulting value to the range $min_i$ to $max_i$.

## REFERENCES


1. Arkin, R.C., Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation, in *Designing Autonomous Agents*, P. Maes, Editor. 1990, MIT Press, Cambridge, pp.105-122.
2. Badler, N.I., C. Phillips, and B.L. Webber, *Simulating Humans: Computer Graphics, Animation, and Control*. 1993, Oxford University Press, New York.
3. Blumberg, B. Action-Selection in Hamsterdam: Lessons from Ethology. in *Third International Conference on the Simulation of Adaptive Behavior*. Brighton, England,1994, MIT Press. pp.108-117.
4. Brooks, R., A Robust Layered Control System for a Mobile Robot. 1986. *IEEE Journal of Robotics and Automation* RA-2.pp. 14-23.
5. Bruderlin, Armin and Thomas W. Calvert. Dynamic Animation of Human Walking. Proceedings of SIGGRAPH 89 (Boston, MA, July 31-August 4, 1989). In *Computer Graphics* 23, 3 (July 1989), 233-242.
6. Galyean, T. A. *Narrative Guidance of Interactivity*, Ph.D. Dissertation, Massachusetts Institute of Technology, 1995.
7. Girard, Michael and A. A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. Proceedings of SIGGRAPH 85 (San Francisco, CA, July 22-26, 1985). In *Computer Graphics* 19, 263-270.
8. Horswill, I. A Simple, Cheap, and Robust Visual Navigation System. in *Second International Conference on the Simulation of Adaptive Behavior*. Honolulu, HI, 1993. MIT Press, pp.129-137.
9. Koga, Yoshihito, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning Motion with Intentions. Proceedings of SIGGRAPH 94 (Orlando, FL, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994, ACM, SIGGRAPH, pp. 395-408.
10. Latombe, J. C., *Robot Motion Planning*. 1991, Kluwer Academic Publishers, Boston.
11. Ludlow, A., The Evolution and Simulation of a Decision Maker, in *Analysis of Motivational Processes*, F.T.&.T. Halliday, Editor. 1980, Academic Press, London.
12. Maes, P., Situated Agents Can Have Goals. *Journal for Robotics and Autonomous Systems* 6(1&2), 1990, pp. 49-70.
13. Maes, P., T. Darrell, and B. Blumberg. The ALIVE System: Full-body Interaction with Autonomous Agents. in *Proceedings of Computer Animation'95 Conference*, Switzerland, April 1995, IEEE Press, pp. 11-18.
14. McKenna, Michael and David Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. Proceedings of SIGGRAPH 90 (Dallas, TX, August 6-10, 1990). In *Computer Graphics* 24, 4 (August 1990), pp.29-38.
15. Minsky, M., *The Society of Mind*. 1988, Simon & Schuster, New York.
16. Raibert, Marc H. and Jessica K. Hodgins. Animation of Dynamic Legged Locomotion. Proceedings of SIGGRAPH 91 (Las Vegas, NV, July 28-August 2, 1991). In Computer Graphics 25, 4 (July 1991), 349–358
17. Renault, O., N. Magnenat-Thalmann, D. Thalmann. A vision-based approach to behavioral animation. *The Journal of Visualization and Computer Animation* 1(1),1990, pp.18-21.

18. Reynolds, Craig W. Flocks, Herds, and Schools: A Distributed Behavioral Model. Proceedings of SIGGRAPH 87 (Anaheim, CA, July 27-31, 1987). In Computer Graphics 21, 4 (July 19987), 25-34.
19. Tinbergen, N., *The Study of Instinct*. 1950, Clarendon press, Oxford.
20. Tu, Xiaoyuan and Demetri Terzopoulos. Artificial Fishes: Physics, Locomotion, Perception, Behavior. Proceedings of SIGGRAPH 94 (Orlando, FL, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994, ACM, SIGGRAPH, pp. 43-50.
21. Tyrrell, T. The Use of Hierarchies for Action Selection, in *Second International Conference on the Simulation of Adaptive Behavior*. 1993. MIT Press, pp.138-147.
22. Wilhelms J., R. Skinner. A 'Notion' for Interactive Behavioral Animation Control. *IEEE Computer Graphics and Applications* 10(3) May 1990, pp. 14-22.
23. David Zeltzer, Task Level Graphical Simulation: Abstraction, Representation and Control, in *Making Them Move*. Ed. Badler, N., Barsky, B., and Zeltzer D. 1991, Morgan Kaufmann Publishers, Inc.