Pavlov: Programming By *Stimulus-Response* Demonstration

David Wolber
Department of Computer Science, University of San Francisco
2130 Fulton St., San Francisco, CA., 94117-1080
(415) 666-6451
wolber@usfca.edu

ABSTRACT
Pavlov is a Programming By Demonstration (PBD) system that allows animated interfaces to be created without programming. Using a drawing editor and a clock, designers specify the behavior of a target interface by demonstrating stimuli (end-user actions or time) and the (time-stamped) graphical transformations that should be executed in response. This stimulus-response model allows interaction and animation to be defined in a uniform manner, and it allows for the demonstration of *interactive animation*, i.e., game-like behaviors in which the end-user (player) controls the speed and direction of object movement.

KEYWORDS
End User Programming, UIMS, Programming By Demonstration, Programming By Example, Animation

INTRODUCTION
A visitor to our planet might deduce that most computer users have the necessary skills to quickly and easily graphical user interfaces (GUIs). First, computer users know what they want: any user of today's popular applications is now quite capable of delivering a detailed (and passionate!) discussion on the strengths and flaws of computer interfaces. Second, most computer users have the mechanical skills required to demonstrate the appearance and behavior of an interface: anyone that has used a drawing editor knows how to draw objects with a computer, click on them, and transform them.

But today's development tools have yet to fully tap the potential of the computer user. Though interface builders like Visual Basic have significantly decreased the time and expertise necessary to build *standard* interfaces, the development of more graphical, animated interfaces is still mostly performed by skilled programmers. This time-consuming and costly development is particularly unfortunate given the exploding demand for computer games, interactive entertainment, and animation in even "standard" interfaces, and the recognized importance of more end-user participation in designing interfaces. There has been some progress: an entire book has been published describing research systems that allow interfaces to be created or extended by demonstration rather than programming [2] ; commercially, tools like Macromedia's *Director* allow non-programmers to develop animation and some interaction.

But none of these systems cohesively combine interactive techniques for specifying end-user interaction, graphical transformation, and timing, the three primary ingredients of an animated interface. *Director* is powerful for specifying transformation and timing, but designing simple interaction requires some programming, and more complex interaction requires an expert. The Programming By Demonstration (PBD) systems in [2] present powerful techniques for specifying transformation and some interaction, but do not provide the timing mechanisms necessary for animation.

This paper presents Pavlov, a GUI development system based on the *stimulus-response* model. Stimulus-response provides a cohesive model for demonstrating interaction, transformation, and timing. The model seeks to minimize the cognitive dissonance between concept and design by allowing designers to demonstrate the behavior of an interface exactly as they think of it: "When I do A, B occurs", or "two seconds after the start of the program, this animation begins." Beginning with a blank *target interface*, tabula rasa if you will, the designer uses a drawing tool to draw the interface, then uses the same tool and a clock to demonstrate stimulus-response pairs. In essence, the designer teaches the system in a way that is intuitive to humans:

> The basic physiological function of the cerebral hemisphere throughout the subsequent individual life consists in a constant addition of numberless signaling conditioned stimuli to the limited number of in-born unconditional stimuli, in other words, in constantly supplementing the unconditioned reflexes by conditioned ones.
>
> Ivan Petrovich Pavlov

CHI 96 Vancouver, BC Canada
© 1996 ACM 0-89791-777-4/96/04..$3.50

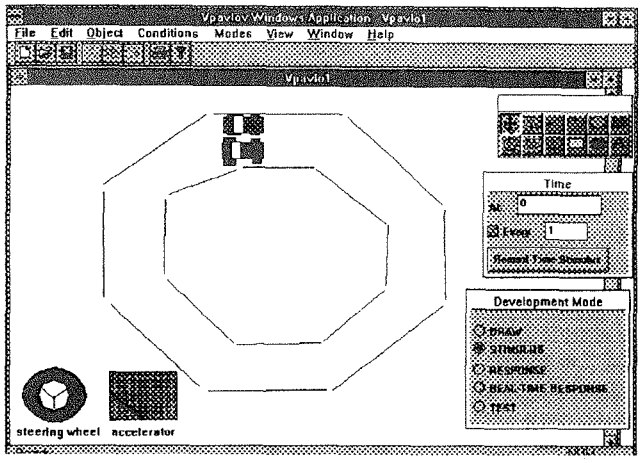

Figure 1. Pavlov Development of a Driving Simulator

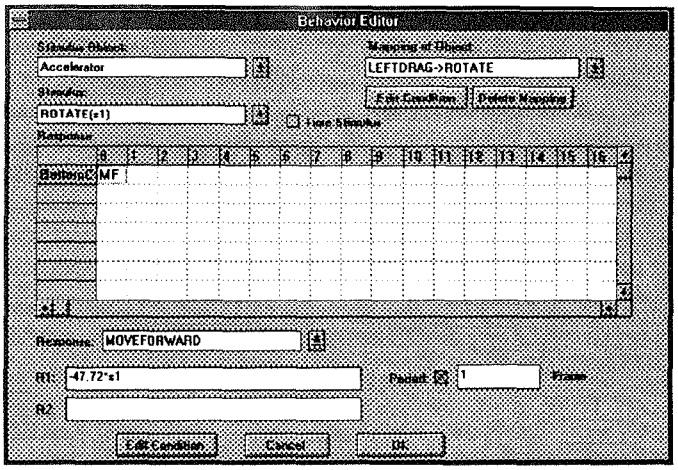

Figure 2. The Pavlov Editor

This stimulus-response model was first used in the author's DEMO system [13] to demonstrate *non-animated* interface behaviors. The model has been extended in Pavlov so that an interface can be taught about time, periodic activity, and the inherent *direction* of some objects. These extensions allow animation as well as interaction to be designed *within the stimulus-response framework.* It is this intersection between animation and interaction, not animation per se, that distinguishes Pavlov from other PBD systems. Because of it, designers can demonstrate most behaviors that combine interaction and animation, including game-like behaviors in which the end-user (player) controls the speed and direction of object movement.

A DRIVING SIMULATOR EXAMPLE

In the driving simulator, the top car begins moving when the program begins, follows a pre-defined path around the track, then stops near its starting point. The bottom car begins moving only when the driver rotates the "accelerator". Its speed and direction is controlled by the driver (the end-user) manipulating the accelerator and the steering wheel.

Figure 1 shows the Pavlov environment during development of the driving simulator (also see the Video Figure in the CHI 96 Video Program). The basic tools are the drawing editor in the top-right corner, the clock (middle-right), and the Development Mode Palette (lower-right). The designer uses the development modes to inform the system as to whether s/he is just drawing the interface (Draw mode), demonstrating an end-user or time stimulus (Stimulus mode), demonstrating how the system should respond to a stimulus (Response or Real-Time Response mode), or testing an interface (Test mode). The designer uses the clock to demonstrate when an operation should be executed (using the top *At:* box), or if an operation should be executed periodically (the middle *Every:* box). The lower button on the clock, labeled *Record Time Stimulus,* allows the designer to specify that the following responses should be triggered by time.

Another important part of the environment is the editor, shown in Figure 2. This editor displays a textual description of the interface being designed. It serves to provide feedback to a designer as demonstrations are performed, and it allows the designer to modify the behaviors "learned" by the system when necessary. Details of this editor are provided later in the paper.

The first task in creating the Drawing Simulator is to draw the two cars, the road, the accelerator, and the steering wheel. To do so, the designer selects Draw mode from the Development Mode Palette, and makes use of the graphic primitives and grouping mechanisms available in the drawing palette. He also provides names to the drawn objects for later reference.

Next, the designer begins specifying the behavior of the interface. Because s/he wants the two objects shaped like cars to move as cars do, s/he selects each, chooses Object | Set Direction from the menu and enters an angle that defines the *direction attribute* of the particular car. In the simulator, both cars initially face straight ahead on the x-axis, so the designer sets the angle to 0. A vector emanating from its center appears on each car to signify that the car will only move forward and backward in relation to its direction, and must be rotated to change direction. The vector does not appear during execution of the target interface.

The designer is now ready to demonstrate the stimulus-response behavior of the top car. In this case, the stimulus is time: at time 0 (the beginning of execution) the car should begin moving. To demonstrate, the designer selects Stimulus mode, sets the clock *At:* box to 0, and clicks on the Record Time Stimulus button. The system reports that a *time stimulus* has been recorded, and automatically

switches to Response mode. For this behavior, the designer wants to demonstrate a special kind of response, a real-time response, so that mode is selected. The designer then selects the move icon in the drawing palette and drags the car around the track. As s/he drags the car, it doesn't ever move diagonally, but instead moves forward towards its nose, and rotates its base (turns) in order to follow the mouse around the track. After the designer releases the mouse button, s/he sees from the editor that the system has recorded a series of discrete, time-stamped responses, made up of alternating MoveForward and Rotate commands.

Next, the designer enters Test mode to visually test the demonstrated behavior. Immediately, the top car begins moving and follows the path that was demonstrated. The designer knows s/he could edit the recorded responses in the editor to modify the path, but for now s/he is satisfied with the behavior of the top car.

The designer then turns his attention to the bottom car. The bottom car's behavior is not triggered by time, but by an end-user action. Thus, instead of demonstrating a time stimulus, the designer plays the role of the end-user and demonstrates an action. After entering Stimulus mode, s/he selects the Rotate icon in the drawing palette, presses the left-mouse button on the rectangle denoting the accelerator, and rotates it clockwise some amount, say -0.36 radians. The system reports that a stimulus was recorded and automatically switches the development mode to Response. The system also records an implicit stimulus-response descriptor mapping the physical action used to the higher-level operation:

(1) On Accelerator.LeftDrag --> Accelerator.Rotate

At this point, the designer needs to demonstrate that the rotation of the accelerator should cause a response of accelerating the movement of the bottom car. First, s/he enters 1 in the *Every:* box of the clock. S/he knows that this will cause the upcoming demonstrated response to be executed periodically every time frame in the target interface. Next, s/he moves the car some amount, say 17 units. Because the car is a "directed" object, the car's movement is restricted: it can only be moved on the vector defined by its direction arrow (note that in Real-Time Response mode this vector is allowed to change). This restriction is as the designer desired: in response to the rotation of the accelerator, s/he wants the car to *move forward*, not change direction. The system reports the recorded stimulus-response descriptor containing a proportional constant (-47.22 = 17/-.36)

(2) On Accelerator.Rotate(s1)-->
BottomCar.MoveForward(-47.22*s1) At 0 Every 1

Next, the designer enters Test mode to check his work. The top car immediately begins its path. The designer, playing the role of the end-user, rotates the accelerator. The bottom car begins moving, and continues to move even after the designer releases the mouse from the accelerator. As the car leaves the right side of the screen, it reappears on the left. The designer again experiments with rotating the accelerator and notices that rotating it clockwise speeds up the car, while rotating it counter-clockwise slows it down.

The designer is nearly satisfied but thinks the accelerator is a little sensitive. He enters the editor (see Figure 2) and selects "Accelerator" as the stimulus object. The stimulus "Rotate(s1)" appears in the box labeled stimulus, and a single response appears in the first cell of the score row labeled BottomCar. The response box below the score contains the response "MoveForward". Its parameter, as in descriptor (2) above, is -47.22*s1. The period box contains "1". To reduce how much the car speeds up in response to the rotate, the designer changes the proportional factor from -47 to -30.0. (alternatively, the period could have been increased). When s/he re-enters Test mode, s/he is satisfied to see that the accelerator is indeed less sensitive.

The next task is to specify the behavior of the steering wheel so the end-user can control the direction of the bottom car. The designer enters Stimulus mode and rotates the steering wheel. Then, in Response mode, s/he sets the *Every:* box in the clock to 1, and rotates the bottom car. The following descriptor is recorded:

(3) On Wheel.Rotate(s1)-->
BottomCar.Rotate(0.25*s1) At 0 Every 1

To test this new behavior, the designer once again enters Test mode. The top car immediately begins its path. The designer rotates the accelerator to get the bottom car moving, then releases the accelerator and rotates the steering wheel to control its direction. He is pleased to note that s/he was correct to set the *Every:* box before demonstrating the rotation of the bottom car: just like a real one, the car continues to turn if the steering wheel remains rotated from its original setting.

The designer continues to test the interface, and soon realizes that if s/he rotates the accelerator counter-clockwise past its origin, the car begins moving backward. To alleviate this problem, s/he uses the editor to delete the previously recorded accelerator behavior, and then re-demonstrates it. First, in Draw mode, s/he sets the top-left point of the accelerator on the left edge of the enclosing rectangle and chooses *Conditions | Generate* from the menu. Then s/he demonstrates the stimulus of rotating the accelerator. A dialog appears listing a set of graphical

conditions relating the stimulus object to other objects in the interface. The designer selects the condition "Accelerator.Within(EnclosingRectangle)", and the system records a modified version of the originally recorded stimulus-response descriptor (1).

(4) On Accelerator.LeftDrag --> Accelerator.Rotate
When Accelerator.Within(EnclosingRectangle)

The designer proceeds to demonstrate the response of moving the car forward, as s/he did in the first iteration. Afterwards, s/he re-enters Test mode, and is satisfied to see that s/he (playing the role of the end-user) is restricted from rotating the accelerator outside its enclosing rectangle. Since the top-left point of the accelerator begins on the left edge of the enclosing rectangle, there is no way to rotate it counter-clockwise past its origin, so the car cannot move backward.

THE STIMULUS-RESPONSE MODEL

The driving simulator example illustrates many features of the stimulus-response model, including the extensions that allow interactive animation to be defined. This section describes the model in more general terms in order to 1) explain the inferences made by the system in the example, and 2) bridge the gap between the specific and the general, i.e., persuade the reader that Pavlov is useful for designing all kinds of interfaces, not just driving simulators.

An interface is viewed as a stimulus-response machine. Stimuli are either physical actions (e.g., drag the mouse while pressing the left-button), higher level operations, or time. The interface responds to stimuli by executing a set of time-stamped operations. Operations either create, transform, or delete objects. The set of operations includes the primitives found in most drawing editors and one additional primitive, *move forward*. This additional primitive allows an object to be moved while constrained to the vector defined by its direction attribute. Together, the operations offer the base functionality necessary to demonstrate nearly all animated interface behaviors.

The Semantics of Stimulus-Response

The challenge of a stimulus-response development system is to provide clear syntax and semantics for how the designer uses the set of physical actions and operations to demonstrate the behavior of the target interface. The "syntax" of Pavlov is straight forward: the designer changes development modes to inform the system whether his intent is to draw, demonstrate a stimulus or response, or test the interface.

Providing clear semantics is a more challenging problem: the goal is for the system to always record a stimulus-response descriptor that perfectly matches the intent of the designer's demonstration (this is the primary challenge of all PBD systems). Pavlov uses an *explanation-based learning* approach: from a single demonstration of a stimulus-response pair, the system uses domain knowledge and the information provided by the demonstration to record as reasonable a stimulus-response descriptor as possible. If necessary, the designer can then use Pavlov's powerful editing facilities to modify the descriptor.

This simplistic strategy differs from other systems that allow a designer to refine behavior descriptions through multiple demonstrations. Such an *empirical-based learning* approach allows more complex behavior to be specified, but complicates the semantics of the system.

The Semantics of a Stimulus Demonstration

In Stimulus mode, the designer demonstrates the operations the end-user can perform in the target interface. When the designer performs an operation in this mode, Pavlov records 1) a stimulus-response pair mapping the physical action used to the operation that was executed, and 2) the first part of a second stimulus-response pair that will eventually map the operation to one or more operations demonstrated as the response.

Mapping the physical action to the operation is important because the drawing palette used during development to demonstrate operations does not appear when the target interface is executed (Test mode). In Test mode, the end-user can only use the operations that the designer has explicitly demonstrated as stimuli, and can only access those operations using the physical action (mouse button, auxiliary key) used in the demonstration. For example, in the driving simulator the end-user can only rotate the accelerator by dragging the mouse with the left-mouse button down, because that is how it was demonstrated. The designer cannot manipulate the car directly in any manner, because no such stimulus was demonstrated. This positive example method of specifying the functionality of the system is in contrast to the scheme of [9] in which the designer "freezes" the objects that cannot be manipulated.

The second recording made from a Stimulus demonstration records the high-level operation demonstrated as a stimulus (e.g., Accelerator.Rotate). It is the execution of this high-level stimulus that will trigger the execution of the responses demonstrated in Response mode.

The only complication to the semantics of a stimulus demonstration is that a designer may demonstrate a stimulus on a *representative* object. At run-time, the same stimulus applied to any member of the set represented will trigger the demonstrated response. Dynamically allocated objects, which can be specified by creating an object in

stimulus or response mode (see [13]), are by default marked as representative objects. Pavlov also allows the designer to designate behavior groups, and marks each element as representative of the group (a similar approach is used in [12]).

The Semantics of a Response Demonstration

When the designer demonstrates an operation R on object O in response mode, the system connects a response of the form "O.R (r_1,$r_{2,...}$) when C" to the previously demonstrated stimulus, where each r_i is a function of zero or more stimulus parameters, and C is an optional context for when R should be executed.

The simplest semantic rule is to execute the demonstrated response each time the demonstrated stimulus occurs in the target interface. However, such a simple rule would preclude the designer from demonstrating the context for when an operation should be executed; semantics such as "execute R is response to stimulus S only when the environment is in state s" could not be demonstrated.

By setting a toggle, the Pavlov designer explicitly states if context should be taken into account. If it is, the designer configures the interface into the desired context (or the negation of the desired context) prior to a response demonstration. After the demonstration, the system runs a set of tests to identify graphical conditions describing the state of the interface. The designer is allowed to select one or more of these conditions and combine them with logical operators to define the context for when a response should be executed.

In the driving simulator, a context was defined on the description mapping the physical stimulus "Accelerator.LeftDrag" and the response "Accelerator.Rotate". A context could also be demonstrated so that the bottom car doesn't run into the top one: the designer demonstrates the stimulus of rotating the accelerator, tells the system to identify context, then demonstrates a response of moving the bottom car so that it intersects the top car. When the system identifies BottomCar.Intersects(TopCar) amongst other conditions, the designer selects it and negates it, and the following behavior is recorded:

(5) On Accelerator.Rotate(s1)->
BottomCar.MoveForward(47.22*s1) At 0 Every 1
When Not (BottomCar.Intersects(TopCar))

In general, there are many true conditions concerning the state of the interface. To reduce the number of conditions listed for the designer, the system only identifies those conditions relating the response object and all other objects in the interface. Because the stimulus object has also been signified as important, relationships found concerning it and the response object are shown at the top of the list of found conditions. When necessary, the designer can use the editor to specify a condition not identified by the system.

A second complication to the response semantics is similar to that discussed in the stimulus section: if the demonstrated response object is representative of a set, the response is applied to the entire set during execution (or a subset, as defined by a context conditional [13]).

A third complication to the response semantics concerns determining the response parameters. The simplest solution is, of course, to execute R during execution with the same parameters as in the demonstration. This is the best solution when the corresponding stimulus has no parameters (e.g., the stimulus is a button click and the response is a move(x=5,y=7)). However, for stimuli that do have parameters, it is often the case that the reaction is proportional, i.e., the response parameter(s) are proportional to the parameters of the stimulus. For instance, the car in the Driving Simulator is rotated an amount proportional to the amount the steering wheel is rotated. Thus, when such a stimulus-response is demonstrated, Pavlov infers proportional constants $C_i = r_i/s_i$ that relate each of the stimulus and response parameters. When the stimulus $S(s_1,s_2,...)$ occurs during execution, the response $R(s_1*C_1, s_2*C_2)$ is executed.

The formula for R illustrates that the system infers the first parameter of the stimulus to be related to the first parameter of the response, the second to the second, and so on. The basis of this inference is that most interface operations either have a single parameter or they have two parameters denoting x and y coordinates, so in practice the corresponding stimulus and response are often related. As with conditionals, the editor can be used to modify the response formulas recorded.

EXTENSIONS FOR ANIMATION

Systems for demonstrating animation have existed for over twenty-five years [1]. Pavlov's contribution is the integration of animation demonstration with the stimulus-response model for defining interaction.

An animation path can be demonstrated with a real-time response demonstration, with a series of time-stamped response demonstrations (the editor can be used for in-betweening), or with a periodic response. Like any other response, an animation path can be triggered by any kind of end-user or time stimulus.

When a designer demonstrates the transformation of an object in real-time response mode, the system records a

series of time-stamped operations. Because operations are recorded instead of picture frames (as in *Director*), the recorded path is not constrained to a particular starting point or object. Thus, reuse is facilitated. The mechanism is also slightly more general than in systems such as *Director* because any operation, not just move, can be demonstrated in real-time.

A Notion of Direction

An important contribution of Pavlov is that a designer can demonstrate animation in which the end-user not only initiates movement, but accelerates it and changes its direction. Though such behavior is the primary activity in many game-like applications, there has been little research in this area, and commercial systems such as *Director* require extensive programming to develop this part of an application.

From struggling with how to allow game-like behavior to be demonstrated, the following observations were made: In many games, one input control (e.g., steering wheel) is used to control direction, and a different control (accelerator) to control speed. Also, many objects do not move in an arbitrary manner, but are restricted to moving forward and backward, and must rotate their base to turn. From these observations it became clear that the standard Move(x,y) operation in Pavlov's drawing editor is not sufficient for the demonstration of movement because it specifies both a distance and an absolute direction.

To solve the problem, a notion of direction was added to the stimulus-response model. Designers can set a *current direction* attribute for an object that is displayed during development. The direction attribute makes it possible for the designer to demonstrate a *MoveForward(d)* operation. This operation causes an object to move forward (or backward, if d<0) in the direction it is facing. Thus, it is much better suited for the demonstration of acceleration than Move(x,y).

Periodic Responses

The notion of a periodic response is also useful in demonstrating acceleration. In the driving simulator, when the end-user rotates the accelerator, the car should begin moving and *continue to move*, even after the end-user releases the mouse. In Pavlov, the designer explicitly specifies continuous movement by setting the *Every:* box on the clock before the demonstration of a forward movement. In essence, when a "MoveForward (d) Every t" is demonstrated, the system infers that the object should move forward at a speed of d/t.

The following run-time rule follows from these semantics: the execution of successive periodic MoveForward operations on the same object results not in two alternating and possibly opposite actions, but in a single action combining the magnitudes of the operations. For example, in the driving simulator, when the car is already moving at 4 units/frame and the end-user rotates the accelerator again, say back towards the origin, it causes a response of MoveForward (-1) units/frame. This second operation is combined with the existing one so that the car slows down to 4-1=3 units/frame, instead of alternating between moving forward 4 units, and backward 1 unit.

The system only uses these semantics for periodic Move-Forward operations. For other operations, successive periodic responses will execute in tandem. Thus, using two periodic, regular Move demonstrations, the designer can demonstrate that an object move back and forth, such as in an *animated move icon.*

An alternative method of demonstrating acceleration has also been added to the Pavlov environment. After demonstrating a stimulus that should cause the acceleration, the designer enters *real-time* response mode and moves an object forward at the desired speed. Generally, a real-time response is used to demonstrate a fixed animation path as a response to a simple button-click or time. When a real-time Move Forward is demonstrated as a response to a transformational stimulus (one with parameters), the system does not record a series of discrete time-stamped operations as usual, but instead records a single periodic operation. The distance parameter is computed by dividing the total distance of the demonstrated movement by the time of the movement (d/t), and the period is set to 1 (ms).

The advantage of this scheme is that the designer truly demonstrates the speed of the movement; the disadvantage is it complicates the semantics of the system. A more thorough analysis will be provided after more feedback is gathered from users.

THE PAVLOV EDITOR

An important aspect of a PBD system is how a designer edits the behaviors inferred by the system. Pavlov's editor borrows from *Director* by providing a time-line view of activity (a score). However, because interaction is emphasized, Pavlov provides multiple timelines: one for the events that occur without an end-user stimulus, and one for the events triggered by each end-user stimulus that was demonstrated. This method of organizing events by stimulus significantly eases the editing task compared to the single score editors found in most animation systems.

Pavlov's editor, shown in Figure 2, can be viewed simultaneously with the main development window. In order to view the operations that occur in response to a particular stimulus, the designer selects an object and a

particular stimulus in the top-left list boxes. To view the operations that occur without an end-user stimulus, the designer selects the "Time Stimulus" check box to the right of the stimulus list.

The objects that respond to the listed stimulus are shown in the rows of the score. The designer can select a particular response in a cell, and the inferred response parameters appear in the edit boxes labeled R1 and R2. Any expression consisting of constants, stimuli parameters, and system-supplied object attribute functions may be entered as a response parameter. In essence, editing behavior formulas is very similar to entering a formula in a spreadsheet.

IMPLEMENTATION

At the beginning of execution (Test mode), the time stimulated operations defined in the target interface are placed in the *execution list* with their respective time stamps. For each end-user stimulus that occurs, the *sr processor* traverses the selected object's stimulus-response list to find the response operations associated with the stimulus. These responses are copied into the execution list with a time-stamp of $t + t_r$, where t is the time the stimulus occurred (the current time), and t_r is the recorded time-stamp of the response (which is relative to the stimulus). When the system is not processing end-user stimuli in this manner, the execution list is traversed and all operations whose time-stamp is less than the current time are executed. A non-periodic operation is removed from the execution list immediately after execution; a periodic operation is left in the list, with its time-stamp incremented by the size of its periodic interval. In either case, the executed operation is sent as a stimulus to the sr processor, so a chaining of events can occur.

Though the scheme does not guarantee that operations will be executed before or on their time-stamp, in practice it provides visually acceptable performance even for interfaces with lots of interaction and concurrent animation (Pavlov runs on a 486 PC).

RELATED RESEARCH

Rehearsal World [5] and *Peridot* [7] were early PBD systems that inspired the stimulus-response framework. The first systems to allow direct graphical demonstration of a full range of stimuli and responses were *DEMO* [13] and *Marquise* [8]. *DEMO* introduced the stimulus-response model and a technique for demonstrating dynamically created objects, while *Marquise* focused on the demonstration of graphical editors, including those with palettes and modes.

DEMO II [3] and [4] are stimulus-response systems that allow the designer to perform multiple demonstrations of the same behavior to refine the system's inferences. [4] uses multiple examples to make sophisticated inferences concerning response parameters-- inferred formulas may depend on attributes of arbitrary objects as well as stimulus parameter values. *DEMO II* uses multiple examples to refine inferences concerning the context for when a response should be executed.

Pavlov is the first *stimulus-response* system to focus on animation, though there are a few PBD systems not based on stimulus-response that allow some animation to be demonstrated: *KidSim* [12] and *Agent Sheets* [10] use graphical rewrite rules to allow designers to demonstrate the context for when an operation should be executed. The systems are powerful for creating non-interactive simulations, but the rewrite-rule method of defining context is not integrated with a method of specifying end-user stimuli, so interactive simulations cannot be designed without coding; *Dance* [11] allows the demonstration of animation for the purpose of program visualization. *Chimera* [6] and *LEMMING* [9] allow interface behavior to be specified with multiple demonstrations of *constraints*, but do not cover time-based animation or acceleration.

Director is representative of the commercial animation systems that provide facilities for both animation and interaction design. These systems allow animation to be designed quickly and easily using a combination of frame-by-frame animation, in-betweening, and real-time recording. These systems also allow sound and video to be linked into presentations, and provide a range of features for creating special effects such as *slow-in/slow-out*, *motion blur*, and *squash and stretch.*

Though powerful for defining animation, these systems do not provide a PBD method of defining interaction. The systems all allow button-click triggered animation to be defined in a relatively simple manner. However, more complex stimulus-response behaviors, such as the steering-wheel and accelerator controlled animation in the driving simulator, require expert-level programming.

A second difficulty in defining interaction with animation systems is that they are based on a single-score editor: all the animation sequences of an application are shown on a single time-line. Though such a score is sufficient for non-interactive animation (which was its original purpose), it is too unstructured for applications with interactive as well as time-stimulated animation. Like the programs written before the advent of structured programming (sub-procedures), the designer is forced to program control, i.e., where one animation ends and another begins, using *goto* statements. For complex applications with lots of movement and interaction, the result is a *spaghetti score.*

The multiple score scheme in the Pavlov editor alleviates this problem, and allows the designer to edit the different interactive behaviors and animation sequences separately.

LIMITATIONS

The major practical limitation of Pavlov is that interfaces created with it cannot be connected to application code. In the next version, designers will be able to make this connection by 1) demonstrating a function call as a stimulus or response, and 2) calling an application function within a response or conditional formula. Pavlov's single demonstration scheme might also be considered a limitation: more behaviors could be inferred if a multiple demonstration inference engine, such as [4], was integrated. Before doing so, however, we want to study whether the additional inferred operations justify the additional complexity that would be added to the environment. A third limitation is that *acceleration* can be demonstrated for an object that has no pre-defined path, but cannot be demonstrated for an object that must stay on a *fixed path.* In this regard, we are exploring both the use of parametric functions to model some animation paths, and the use of "conductor" objects with special properties [10].

SUMMARY

Pavlov contributes a cohesive model for demonstrating animation and interaction, and innovative techniques for demonstrating interfaces in which the end-user controls both the speed and direction of animation paths. Using these techniques, interfaces like the driving simulator can be created in less than fifteen minutes.

Development is by no means restricted to driving simulators or similar applications. A number of other interfaces have also been developed, including a wide variety of games, a diagram editor with animated icons, and an educational solar system program. We attribute the general usefulness of Pavlov to the generality of the stimulus-response model, and its powerful multiple-score-based editing facility.

Besides increasing the range of PBD, the stimulus-response model provides a very intuitive method for defining interfaces. A usability test was performed with a number of non-technical designers. Subjects were given a manual describing the stimulus-response model, and then were asked to design two interfaces equal in complexity to the driving simulator. Seven of ten were able to create the interfaces within an hour; the three others completed the tasks after asking a few questions (specific questions concerning how to complete the tasks were not allowed). We were extremely encouraged by the results, as well as the enthusiasm the subjects expressed for exploring once the formal tests were completed (though we could get none to salivate!).

REFERENCES

1. Baecker, R., Picture-Driven Animation, *Proceedings of the Spring Joint Computer Conf.*, AFIPS Press, 1969, pp. 273-288.

2. Cypher, A., ed., *Watch What I Do: Programming By Demonstration*, MIT Press, Cambridge, Mass., 1993.

3. Fisher, G., Busse, D.,and Wolber, D.,"Adding Rule Based Reasoning to a Demonstrational Interface Builder,*Proceedings of UIST'92*,Nov. 1992, pp.89-97.

4. Frank, M. and Foley, J., "A Pure Reasoning Engine for Programming By Demonstration", *Proceedings of UIST '94*, Nov. 1994, pp. 95-102.

5. Gould, L. and Finzer, W., "Programming By Rehearsal", *Byte,* v. 9, no. 6., 1984.

6. Kurlander, D. and Feiner, S., "Inferring Constraints from Multiple Snapshots", *ACM Transcations on Graphics*, May, 1991.

7. Myers, B., *Creating User Interfaces By Demonstration*, Academic Press, San Diego, 1988.

8. Myers, B., McDaniel,R.,Kosbie, D.,"Marquise: Creating Complete User Interfaces By Demonstration, *Proceedings of INTERCHI '93*, Amsterdam, April,1993, pp.293-300.

9. Olsen, D., Ahlstrom, B., Kohlert, D., ,"Building Geometry- based Widgets by Example", *Proceedings of CHI '95*, May, 1995, pp.35-42.

10. Repenning, A., "Agent Sheets: A Medium for Creating Domain-Oriented Visual Languages", *Computer*, V.28, 1995, pp. 17-25.

11. Stasko, J., "Using Direct Manipulation To Build Algorithm Animations By Demonstration", *Proceedings of CHI '91*, 1991, pp. 307-314.

12. Smith, D.C., Cypher A., "KidSim: End-User Programming of Simulations", *Proceedings of CHI '95*,May 1995, pp.27-34.

13. Wolber, D., and Fisher, Gene, "*A* Demonstrational Technique for Developing Interfaces with Dynamically Created Objects." *Proc. of UIST '91*, 1991, pp. 221-230.