Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

System for Authoring Highly Interactive, Personality-Rich Interactive Characters

A. Bryan Loyall, W. Scott Neal Reilly, Joseph Bates and Peter Weyhrauch

Zoesis Studios, Newtonville, MA, USA

Abstract

We describe an innovative system for authoring expressive, fully autonomous interactive characters. The focus of our work is creating a system to allow rich authoring that captures as much of the artistic intent of the author in procedural form as we can, and that provides automatic support for expressive execution of that content. The system is composed of two parts: (1) a programming language with unusual language features including concurrency, reflection, backtracking, continuously monitored expressions, and a model of emotion, that was created for the expression of interactive self-animating characters; and (2) a motion synthesis system that combines hand-animated motion data with artistically authored procedures for generalizing the motion while preserving the artistic intent. This system has been used to create over a dozen interactive characters, which have been shown at juried venues, as well as being deployed commercially. We describe how artistic qualities important to interactive characters are encoded and supported using this system, and demonstrate the system with an implemented interactive character.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation I.2.0 [Artificial Intelligence]: Cognitive simulation

1. Introduction

The wealth of details that make Bugs Bunny the compelling personality that he is were created by talented artists who carefully crafted every detail of his motion, with all of the principles of animation, human common sense, knowledge of (cartoon) physics, emotional psychology, social skills and acting ability at their disposal. This human artistic knowledge guided their decisions at every moment of their linear film to bring the character to life. For fully autonomous, interactive characters to approach this level of richness, we must enable skilled artists to craft each of these details *procedurally*, so that the interactive character can have the knowledge it needs to make the right decisions at all of the moments of its interactive "life." This information must be encoded procedurally in the character, because the artist will not be there to help make the decisions when the character is executing, and because the character has to make the right interactive choices that the artist would make if they were there.

Figure 1: *Mr. Bubb, an example interactive character.*

The more interactive and rich the system, the more knowledge-based active elements the character must have to react appropriately. This knowledge doesn't have to be encoded in a general way for any character and any situation, which would be an impossible task for the foreseeable future, but it must be crafted for the specific character and the full generality of the interactive situations which can occur in its world.

In this paper, we present a system to enable the authoring of rich procedural knowledge of interactive characters. This system is composed of two parts: a programming language, called *Gertie*, specialized for the expression of interactive

self-animating characters; and a motion synthesis system, called *the WB*, that allows similar expressive authoring for generalized acts. (Gertie is named for Gertie the dinosaur by Winsor McCay, widely regarded as the first animated character, and the WB is named after warping and blending because of its use of motion reuse techniques.)

We demonstrate the qualities of the system by descriptive, illustrative examples taken from an implemented interactive character, Mr. Bubb. An image of the character and physical environment is shown in Figure 1. The interactive experience is a simple game with Mr. Bubb, similar to one children might play where they take turns hitting a balloon to keep it from hitting the ground. The user controls a hand in the world with a mouse. The game is purposefully simple, to focus the experience on interactions with the personality-rich, emotional character. Excerpts of interactions are shown on the video.

2. Related Work

There has been much work on interactive characters in both computer graphics and artificial intelligence. This includes work in behavioral animation such as the early work of Reynolds [Rey87] and later work by Funge [FTT99] in which the character behavior typically focuses on appropriate group behavior with less support for the expression of individual personality. Work in realistic virtual humans has also been an active area of research with the early and continuing work of Magnenat-Thalmann and Thalmann [MTT87] and Badler [BPW93] as representative examples. Interactive characters for education has more recently been actively pursued. In this work, the expressiveness of the characters is in service to pedagogical goals [RJ99, LZGB99] and typically focuses less on expression of individual personalities than our work. Related work is also pursued in simulation and training (e.g. [GM01]). This work includes strong personality-types, but works in conversational domains rather than dynamic-animation-based domains. The area of embodied conversational agents (e.g. [AR02] is a good representative example) is similar in its de-emphasis of action. Advances being made in these conversational agents may be complementary to our work.

One of the systems that focuses on authoring of expressive personality-rich interactive characters is Perlin and Goldberg's Improv system [PG96]. This system produces compelling motion, especially for background actions that keep the character seeming alive without the appearance of repetition. Their system also provides a behavior authoring system using nested scripts of a constrained subset of English, with the goal of allowing authoring by non-programmers. A side-effect of this is that the scripting mechanism provides less structured support for authoring complex behavior than we focus on.

Another group of systems that supports complex interactive characters is the work of the MIT Media Lab Synthetic Characters Group [BDI*02, Joh95]. Their systems feature models of the mental structures for synthetic agents, including interesting models of synthetic vision and learning. The authoring ability in these systems is not the main focus, causing it to be less expressive than the approach we present. In addition, the main goal of this work is making ethologically correct synthetic animals rather than interactive characters, which leads to very different system choices and capabilities.

Mateas and Stern take an expressive authoring approach to their interactive story system Façade [MS02], however their emphasis avoids dynamic animated interaction by focusing on "dinner-party" interactions with little physical interaction. This has the effect of focusing the system on higher-level cognitive abilities of the agent (such as conversation and dramatic story control), and makes it a complementary line of research with our own.

Commercially, interactive characters are used in computer games, crowd scenes in film production, other entertainment applications, and education, and we believe they are a central component of what will become a new form of mass market entertainment that will ultimately be as popular as film and television are now—truly interactive stories. However, at the current time, these commercial applications typically use simpler and less capable technologies such as motion playback with simple blending.

3. Approach

Because we want to create interactive characters that are continuously making choices and changing their behavior as if the artist were there making the choices for them, the approach we take to building these characters is to encode their behavior and motion procedurally in a language designed for this purpose.

This means that the ideal author is someone who is both a programmer and a character artist. Such people are becoming more common as courses are created in technology-based art and entertainment, as artists are drawn to technology, and as technologists are drawn to the creative outlets that technology affords. At Zoesis we have gathered and are continuing to attract and train such artists.

Although it is impossible to prove such a claim, we believe that for highly interactive interactive characters, there is no other way to build them than by building small "brains" to encode the artist's conception of the character.

4. Structure of the System

The structure of our system is shown in Figure 2. The content that an author creates is shown on the right, and is composed of motivations, goals, behaviors, styles of acting, reactions, emotional expressions, motions, etc. This content is encoded from high-level aspects of the personality to low-level motion. The upper levels are authored in Gertie and low-level motions are expressed both in Gertie and as acts in

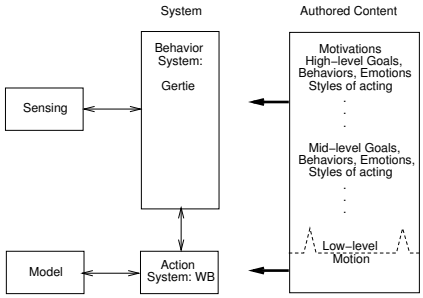

Figure 2: *System structure.*

the WB. The dashed line in the middle of low-level motion indicates this change in authoring language. Originally all low-level motion was authored as acts, but as we describe in Section 5.2, we have found it often to be advantageous to author low-level motion in Gertie. The unevenness of the line shows that there is no fixed point for this change in authoring language, and authors are free mix the two languages as appropriate to their needs. This mixing and the reasons for choosing one versus another are discussed in Section 5.2.

The authored content is used at runtime to bring the interactive character to life. Each frame, Gertie executes the active goals of the character, sensing the world, executing behaviors in service to the goals of the agent, and issuing and aborting parameterized acts in the WB. The WB uses the current set of acts each frame to produce values for the degrees of freedom for the body.

In the following sections we describe the components of the system in detail. While it is impossible to describe all the implementation details due to space, we attempt to describe key techniques by which the system provides support for the central challenges in bringing interactive characters to life. We first describe Gertie, followed by behavior authoring in Gertie, and low-level motion authoring in Gertie. We then describe the WB, followed by discussion and conclusion.

5. Gertie

Gertie is a complete modern programming language with unusual features, and draws on our previous work [Loy97, Nea96, LB93], as well as the work of Sengers [Sen98]. It is designed to allow authors to encode complex details of an interactive character including: the motivations of the character and how those motivations change; the goals, behaviors and styles of action at all of the levels of detail from large goals to details of low-level movements; how these behaviors and goals interact with each other; how the character reacts to the user and unexpected events during these behaviors and goals; and what does the character become emotional about, and how does it express those emotions.

In addition to interactive characters, we encode aspects of our interactive music system and interactive drama control in Gertie, although descriptions of these uses are outside the scope of this paper.

The core of Gertie are the normal constructs common to a modern high-level language. It has a lisp-like syntax and C-like semantics (e.g. there is no garbage collection and data-types are similar to those in C which enables a rich foreign function capability with C and C++). It also has a powerful macro system similar to that found in Scheme.

The unusual language features that are supported in Gertie are: concurrency, reflection, continuously monitored expressions, backtracking, and action and sensing. In the remainder of this section, we describe each of these unusual features, giving examples of their use in the different levels of authoring of an interactive character, and describe additional language constructs that enhance the authorial expressiveness of these features.

Function calls in Gertie are used both to express normal programming as well as goals of the agent. It is often useful to think in both terms when writing Gertie programs and we will use the terms interchangeably. Similarly, function definitions specify the (reactive, conditional, emotionally varying) behavior to execute the goal. The top-level of a character typically has several goals executing concurrently. Some of these are high-level motivations and goals of the character, some structure high-level motivations and goals, and some encode other aspects of the character's mind such as: behaviors to carry out inferences to interpret the complex flow of activity; demons to carry out physical reactions; etc.

To explain the unusual features of Gertie as a programming language and illustrate how it is used to encode behaviors, consider the following illustrative pseudocode of a portion of Mr. Bubb. (Note that details have been elided, such as parameters to the functions and some annotations, in order to focus on the control flow and expressive features of the language being described. Further details are described in following sections.)

```
(defun playGame ()                                              (1)
  (persistent                                                   (2)
    (seq (bubbTurn) (userTurn)))                                (3)

(defun bubbTurn ()                                              (4)
  (seq                                                          (5)
    (concurrent                                                 (6)
      (persistent (priority-mod +1) (deferToUser))              (7)
      (moveIntoPosition ball))                                  (8)
    (hitBall ball))))                                           (9)

(defun userTurn ()                                             (10)
  (concurrent                                                  (11)
    (annotate (optional) (encourageUser))                      (12)
    (recognizeHit)                                             (13)
    (annotate (optional) (assistIfNeeded))))                   (14)
```

Complex structures of behavior, action and arbitrary computation can be created by nested sequential and concurrent expressions. For example, the structure of the game is expressed as a repeating (`persistent`) sequence (`seq`) of the respective turns (line 3), with `bubbTurn` composed of

a nested `seq` and `concurrent`. Additional structure can be expressed by annotations within `concurrent` expressions. The author can specify that some of the component expressions are optional, or that a smaller number than the entire set are needed to complete in order for the `concurrent` expression to complete. For example, in `userTurn` (in lines 10-14) both `encourageUser` and `assistIfNeeded` are annotated as optional which causes them to be executed but not be considered for success of the userTurn. The behavior will succeed when `recognizeHit` succeeds.

Behaviors are grounded in acts and sensing expressions. Both can be viewed as normal functions. Acts are executed by the WB; Gertie issues the act to the WB and waits on that thread for the act to succeed or fail. (Note that other threads continue concurrently.) If the behavior aborts the act for any of the reasons described below, an abort call is sent to the WB. Typically Gertie issues several overlapping acts each second and aborts a fair number over time. Sensing expressions are task-specific functions that return simple information about the world. Complex information is infered by the patterns of these simple sensations and their effect in context of structured behaviors.

Priorities and mutual-exclusion information can be added to functions and acts. Any two functions, two acts, or a pair of a function and an act can be marked as mutually exclusive (using the `defconflict` primitive), and will be automatically prohibited from executing at the same time by Gertie. The one that has a lower priority is suspended until the higher priority one completes. The suspension could happen before the lower-priority act or function starts executing or at any time during its execution. For example, Mr. Bubb's author wanted him to be socially aware and considerate of others' desires. One detailed instantiation of this is that while hitting the ball, Mr. Bubb attempts to allow the user to hit whenever they want. This is expressed by creating adding another behavior, `deferToUser`, to run concurrently with the main behavior `moveIntoPosition`. `DeferToUser` is a simple demon that monitors whether the user is trying to hit the ball and causes Mr. Bubb to stand back and watch him if so. The `(priority-mod +1)` annotation causes `deferToUser` to execute at a higher priority than its siblings, and elsewhere the declaration `(defconflict deferToUserBody moveIntoPosition)` specifies that the body of `deferToUser` is mutually exclusive with `moveIntoPosition`. The result is that while executing `moveIntoPosition`, Mr. Bubb continuously monitors the user in parallel due to the `concurrent`. Anytime the demon fires, `deferToUserBody` executes causing `moveIntoPosition` to be automatically suspended, which has the effect of causing Mr. Bubb to stop and watch whenever he thinks the user is trying to hit. This can happen multiple times, because the `persistent` annotation causes `deferToUser` to reset after executing.

Failure is an extended concept in Gertie designed to allow the author to express what to do when an act or behavior fails. The mechanisms for handling failure are similar to exception handlers in traditional programming languages. There are four constructs that can handle a failure of a Gertie expression: `ignore-failure`, `persistent`, `persistent-when-fails` and `one-of`. `Ignore-failure` aborts the expression and treats the failure as a normal return. `Persistent` and `persistent-when-fails` both reset the expression, causing it to be available to be executed again. A `one-of` expression is a more complicated construct. A simple example is presented here. (Note that summaries of larger computation are given as descriptions in angle brackets.)

```
(defun hitBall (style)
  (one-of
    ((precondition <on left side>) (hit-with-left-arm style))
    ((precondition <close>) (hit-with-head style))
    (hit-with-both-arms style)))
```

A `one-of` expression is a way to author alternatives to accomplish a task. At execution, it will choose one of its component expressions to execute using preconditions and partial-ordering information provided by the author. Above, two of the alternatives have a precondition, and no partial order is provided. If the preconditions and partial order does not uniquely specify an alternative at execution time, one is chosen randomly from the top candidates. When a failure is handled by a `one-of` the executing alternative is aborted, and another is chosen to be executed, excluding any that have already failed. For example, the above function might first try to hit the ball with its head, and if that fails (perhaps due to an act failing), the `one-of` would choose `(hit-with-arm style)` or `(hit-with-both-arms style)`. If no expression is eligible to be chosen (due to failures or false preconditions) the one-of expression fails. The result is that the character tries any ways he knows to accomplish a task, failing when his options are exhausted.

Gertie includes a simple form of reflection. Any code can query the status of a function or act, which could be one of: executing, available for execution but not executing, suspended, completed successfully, or failed. Reflection is used by an emotion system that is built in Gertie to make it easy to author what a character becomes emotional about, and how they express those emotions, as well as by character code to coordinate between different goals and behaviors.

The emotion system itself automatically generates emotions based on this information, along with the author's notations of which goals are emotionally important. (For example, Mr. Bubb cares if he is playing well, so his `hitBall` function is annotated as being emotionally important.) When a goal that has been marked as emotionally important fails, the emotion system uses reflection to notice this failure, and automatically causes `sadness`. If the character had inferred a cause of the failure, the system causes the creation of an `anger-at` emotion; and if the character infers that an executing act is likely to fail, the emotion system creates a `fear` emotion. (A more complete exposition of the causes and types of emotions that can be automatically generated in this way can be found in [Nea96].) The created emotions

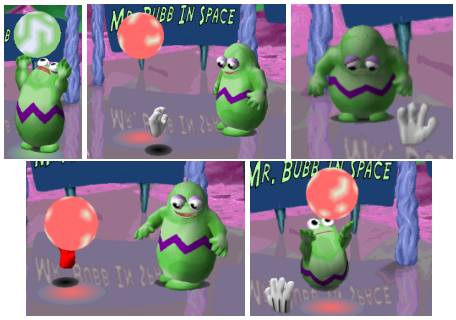

Figure 3: *Interactions with Mr. Bubb. He becomes happy when you play well together and can become sad if you hog the ball or if he plays badly. He is socially aware, and tries to give you space and encouragement to hit, and will try to hit for you when he thinks you can't reach it.*

Property Supported	Example from Mr. Bubb
high level goal authoring	play with user, mope, and play alone
personality-specific, reactive behavior	move to hit the ball unless user is trying to hit
expressive motion authoring	sneeze, stretch and hit motion
complex understanding	user hit for me to help vs. user hogging the ball
emotional causes	playing badly produces sadness
emotional expression	move slower, choose to mope
general awareness	recognize accidental hit
social conventions	getting out of user's way
reactive staging knowledge	if time, move & face camera before hitting ball
physical reactions	react to being hit by ball; slipping on the ice
showing changes in thought	when *accidently* make a good hit, show surprise

Figure 4: *Character properties supported in Gertie.*

are automatically summarized and decayed over time by the emotion system, and the current emotional state is available to all of the executing Gertie code, so that it can continually adjust its choices based on the emotions of the character.

Note that these inferences for cause and likelihood of failure, etc. don't have to be correct. Some characters might always blame themselves, or jump to hasty conclusions based on little evidence, while others may make careful, reasoned inference. In addition, the inference process is written in Gertie and has access to the same emotional state information. This makes it easy to author a character who increasingly blames others as they become anxious or similar effects.

The last feature to describe in Gertie is continuously monitored expressions, which are used in interactive characters to express reactivity to the world (especially including the user) or to unexpected internal events. They can be added to any expression in Gertie, and they come in two versions, `succeed-when` and `fail-when`, that as a group we call whens. During the execution of the expression to which they are attached, the condition of the `when` is continuously evaluated. At the point it becomes true it aborts the attached expression, and causes it to return normally or fail depending on the type of `when`. An optional `replace-with` expression can be present in a `when`. If present, it is executed when the `when` triggers. Example uses are given below.

5.1. Behavior Authoring in Gertie

Now that we have given an overview of Gertie, let us turn to a description of how it provides support for authoring of interactive characters. Figure 3 shows representative scenes from an interaction with Mr. Bubb, and Figure 4 lists properties of interactive characters that are supported by Gertie. For each property, an illustrative example is given from Mr. Bubb.

To illustrate how rich, interactive behavior can be encoded in Gertie, let's further consider the basic structure of Mr. Bubb's turn in the game. At the start of Mr. Bubb's turn, the user has just hit the ball. His basic goal is to possibly give a reaction to the User's hit (particularly if it was a good hit), move into position (being reactive to both the ball and the user) and then hit the ball. This is expressed in the following more detailed pseudocode for `bubbTurn`.

```
(defun bubbTurn ()
  (seq
    (if (nice-hit)
        ((succeed-when <ball is far relative to its height>
                  (replace-with (small-react-to-nice-hit)))
         (react-to-nice-hit)))
    ((succeed-when (or <in position for ball>
                       <ball is too low>))
     (concurrent
       (persistent (priority-mod +1) (deferToUser))
       (moveIntoPosition ball)))
    (hitBall ball)))
```

This behavior is a sequence with three parts. The first expression is a conditional that determines whether Mr. Bubb thinks this is a nice hit. If so, the body is executed: a `succeed-when` continuously monitored expression modifying the main behavior `react-to-nice-hit`. The `replace-with` expression of the `succeed-when` specifies a Gertie expression to execute instead of the main behavior. So, normally, the `react-to-nice-hit` behavior would execute to completion performing an acknowledgment to the user that Mr. Bubb thinks she made a nice hit. If the ball is too far away relative to its height, the `react-to-nice-hit` is aborted by the `succeed-when`, and replaced with the shorter reaction `small-react-to-nice-hit`. Also, since this is a continuously monitored expression, if the ball moves too far away (e.g. because the wind blows it away) during the bigger reaction, it will be aborted and Mr. Bubb will transition to the smaller reaction.

The core of the next expression moves Mr. Bubb into position for hitting while being polite to the user, as described previously. The `succeed-when` around this expression causes the behavior to end whenever Mr. Bubb is already in position (or if he becomes opportunistically in position, e.g. because of the wind) or when the ball is too low (and his only chance to hit it is to stretch out and try to hit it).

Gertie supports staging knowledge in interactive characters as the term is used in theater, that the actor reasons about what the audience can see and moves himself to enable the

Acts	Low-Level Behaviors
openMouth	dissapointedGesture
squashBody	tenseMouth
headLean	wringHands
armFlex	cringe
moveEyelids	frustratedFace
say	sneeze
lookAt	
RecoverFromFaint	

Figure 5: *Representative acts and low-level behaviors.*

Figure 6: *An execution of the sneeze behavior.*

best presentation. Good actors do this in a way that doesn't interfere with their main action or hamper believability. This can be expressed in Gertie by including behavior to, e.g., turn toward the camera or move to an advantageous position as part of the behavior. These behaviors can be written similarly to the reaction to a nice hit above, to allow them to adapt to the time available and other aspects of the ongoing behavior so that they integrate seamlessly. Many of the Mr. Bubb behaviors include adaptive staging.

Showing changes in thought is one of the main ways of bringing a character to life, as has been repeatedly described in animation and acting. Such changes can often be easily locally expressed in Gertie, for example `succeed-when` and `fail-when` expressions often encode conditions under which a character stops executing a given goal and behavior and starts executing another. The `replace-with` clause in each `succeed-when` and `fail-when` executes whenever these changes in goals occur. By adding appropriate behavior to these clauses to express the change in thought we are able to give our interactive characters the ability to show these changes in thought with little additional code.

In order to express the emotional life of the character, an author encodes the goals that are emotionally important to the character, and code to generate additional knowledge needed for the generation of particular emotions, as described above. In Gertie, often these inferences can be easily expressed by adding small expressions at key points in the code. For example, in the `deferToUser` behavior described above, Mr. Bubb is already computing that the user is in his way preventing him from getting to the ball. If Mr. Bubb misses the ball while `deferToUser` is executing, he blames the user for the goal failure, which causes anger toward the user as well as sadness to be created.

5.2. Motion Authoring in Gertie

One of the important properties of Gertie is that it is a powerful language for authoring *motion* as well as behavior. In fact, in most of the recent interactive characters we have built with our system, the authors have chosen to use a small set of simple, short duration acts in the WB, and have chosen to build most low-level motion in Gertie using that small set of WB acts. An illustrative list of typical acts in the WB and low-level motion built in Gertie is shown in Figure 5.

It is important to point out that this is not due to the WB being incapable of creating longer, expressive acts. It is often used for this (e.g. the slide on ice, crash into object, and faint motions, which are all shown on the video, are each long, expressive acts built in the WB), and authors are free to mix either method as they choose. Rather, this choice is due to the advantages that building actions in Gertie provides. These advantages are: (1) for many motions it is easier and faster to create the desired motion by composing lower-level acts than by creating new animation and generalizing it with the WB; (2) using Gertie, the author is able to achieve some effects that would be difficult or impossible in our action system or other action systems that typically reason in terms of animation control curves; and (3) the author's intentions with respect to complex interactions between the act and other acts and behaviors can be easily expressed when the motion is created in Gertie.

To illustrate these properties, we describe a typical expressive motion written in Gertie: a sneeze from Mr. Bubb. The animation from an example execution of the sneeze is shown in Figure 6 and in the accompanying video. The motion is typically between 4.4 and 6.4 seconds long.

There are seven acts that are used to create the `sneeze` behavior (the first seven acts in Figure 5), and the structure of the behavior is shown here. Each of the function and act calls have several arguments that we omit in order to focus on the expressive structure of the behavior. The main arguments to the sneeze behavior itself are `a, s, r` for the desired timing of the attack, sustain and release parts of the motion.

```
(defun sneeze (a s r)                                          (1)
  (concurrent                                                  (2)
    (seq (with-timeout a                                       (3)
           (sneeze-anticipation))                              (4)
         (sneeze-payoff))                                      (5)
    (optional (succeed-when <prone>)                           (6)
       (seq (wait-for) (squashBody) ...))                      (7)
    (optional (priority-mod -10500)                            (8)
      (concurrent                                              (9)
        (seq (armflex right) ...)                              (10)
        (optional (seq (armflex left) ...))))                  (11)
    (optional (seq (moveEyelid left) ...))                     (12)
    (optional (seq (moveEyelid right) ...))                    (13)
    (optional (seq (wait-for ) (lookAt ) (lookAt )))))         (14)

(defconflict openMouth say)                                    (15)
```

```
(defun sneeze-anticipation ( )                                  (16)
  (concurrent                                                   (17)
     (seq (openMouth) ...)                                      (18)
     (persistent (one-of (say almost-sneeze1) ... )             (19)
     (seq (headLean) ...)))                                     (20)

(defun sneeze-payoff ( )                                        (21)
  (seq                                                          (22)
    (headLean ) ...                                             (23)
    (concurrent                                                 (24)
      (seq (headLean ) ...)                                     (25)
      ((priority-mod +10000)                                    (26)
       (concurrent (armFlex left) (armFlex right)))             (27)
      (one-of (say sneezeWord1) ...))))                         (28)
```

The basic structure of the motion is accomplished by nesting `concurrent` and `seq` expressions to create the appropriate structure of acts. The main `concurrent` in the `sneeze` has six clauses (lines 3-14) that respectively specify parallel motion for the: head and mouth (lines 3-5); shoulders (6-7); arms (8-11); left eye lids (12); right eye lids (13); and eyes (14). Timing of the motion is specified by parameters to the acts (omitted in the code structure above), `with-timeout`, `wait-for`, and `optional` expressions. `With-timeout` is a primitive that causes the enclosing expression to succeed after the specified amount of time (in line 3 the '`a`' parameter, which specifies the desired attack duration for the sneeze motion). `Wait-for` is a primitive that introduces a delay for the specified amount of time. `Optional` annotations on clauses of a `concurrent` are used to make those clauses be aborted whenever all of the non-optional clause complete. In this case, all but one of the clauses of the `concurrent` are marked as `optional` so when the unmarked clause completes the `concurrent` completes, causing all other behavior in the `concurrent` to end at the same time.

The `sneeze-anticipation` is an example of generalized motion that would be difficult to create as a single act in a typical motion system, including ours. The main part of the `sneeze-anticipation` is a randomized sequence of mouth gesticulations (created by `(seq (openMouth) ...)`) and "almost-sneezing" vocalizations with associated lip-sync (created by the `say` act). There are a number of ways to create such a randomized sequence in Gertie. In this case, the author chose to express it as two artist-created sequences that are randomly mixed together. The first is a fixed sequence of mouth gesticulations done with a `seq` of `openMouth` acts with fixed parameters (line 18); and the second is an infinite random sequence of "almost sneezing" vocalizations created by line 19. The infinite sequence is created by the `persistent` expression, and the `one-of` expression causes a different authored `say` invocation to be chosen each time. The choices in the one-of all use the `say` act with different parameters (authored and randomized). Because `openMouth` and `say` conflict and are equal priority, Gertie will intermix the two sequences automatically, resulting in an authored pseudo-random sequence.

Gertie automatically layers acts and behaviors that arise from concurrency in the character. By expressing low-level motion in Gertie rather than as a single act in the WB, the author can also specify finer-grained information about how it should interact with other behaviors and acts. For example, the `priority-modifier` expression on line 8 lowers the priority of the enclosed expression that controls the arms. This lower amount causes the arm motion to have higher priority than the behavior that moves the arms randomly as a background behavior, but lower priority than the behaviors that move the arms intentionally (such as the behaviors to hit the ball). On line 26, however, the arm motion at the instant of the sneeze is annotated to have a priority that is higher than the intentional movements. This encodes the artist's conception that Mr. Bubb should be able to concentrate enough to still move his arms to hit the ball while he is sneezing, but at the instant of the sneeze he cannot; the sneeze takes precedence.

Other types of this fine-grained authoring are demonstrated by lines 6-7 which show how a `succeed-when` expression can be used to reactively prevent part of the motion from executing under certain conditions. In this case the `squashbody` act (which performs a shoulder motion) will not happen if the character is lying down, and will stop if the character lies down during the sneeze motion.

As we mentioned above, low-level movements can be authored in either Gertie for the above advantages or in the WB. Typically authors choose to use the WB where the motion would be faster to create as an act, and the additional level of sophistication in variation of the motion and interaction with other acts isn't needed. In Mr. Bubb, for example, the behavior of losing control and sliding on the ice is expressed in five large acts: slide, sitSlide, crash, faint and recoverFromFaint. Each of these would have been more expensive to create in behavior (in large part because they would have required new primitive acts, such as a sitting act, to be created), and they each take over nearly the entire body, so complex interactions with other acts is not as needed.

6. The WB

A character built in Gertie typically sends several overlapping acts each second to be executed and aborts some of the previously issued acts. This stream of dynamically changing, overlapping acts is the natural result of a character that is highly reactive to a rich unpredictable world.

To handle this high interactivity, for personality-rich interactive characters, an action system needs to satisfy four goals: (1) it should allow for dynamically changing layered motion, e.g. the mind should be able to decide to reach to grab something while walking and doing other acts and be able to change the grab to a wave at any point; (2) a side-effect of the first goal above is that acts need to be able to transition to other acts at any nearly any point—if transitions are limited, then interactivity is proportionally limited; (3) it needs to produce authorable generalized motion with artistic quality that approaches hand-created motion; and (4) it needs to be realtime. In the remainder of this section, we briefly survey previous motion control

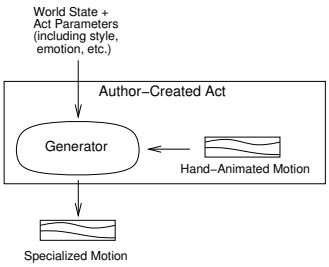

Figure 7: *Mechanism for generalized motion.*

Figure 8: *An execution of a slide act.*

systems, especially with respect to these goals, and then describe our novel motion control system.

A most common motion control approach is verbs and adverbs by Rose [RCB98], with related versions by Johnson [Joh95], and Downie and Blumberg [BDI*02]. These systems interpolate between different animations of the same motion performed in different styles. Transitions from one action to another is done by a transition graph. The expressiveness of individual motions is large, however, we did not believe that interpolation alone would enable authors to capture the full range of desired generalization in both parameters and style, and if it is possible, the number of example animations required is proportional to the desired generality which makes the authoring cost too large. In addition, the need for (near) arbitrary transitions is not well supported by this approach.

Many systems produce compelling, expressive motion, but are not real-time, for example [Gra00, HP97, WK88].

Motion graphs (e.g. [KGP02]) are a new set of algorithms that create a reasonably dense graph of transitions given a rich set of example animations. This approach is compelling, however it currently only works on whole body animation not satisfying our first goal, and it would require a large number of animations to get a sufficiently broad range of expressive motions.

Finally, Perlin and Goldberg's approach to generalized action [PG96] is the closest to satisfying our goals. Their approach is real-time, produces expressive motion, supports transitions, and supports composable actions for parts of the body. The reason we did not adopt Perlin and Goldberg's motor control system is that we believed that it would not be easy enough to author and achieve the full range of generalized action that we need.

Our approach to allowing the authoring of generalized motion is shown in Figure 7. Each act is a combination of author-created hand-animated keyframe curves, which we call the *gesture* for the act, and an author-created procedure, the *generator*, that expresses how to generalize the motion. This approach is similar to Grassia's *motion models* [Gra00].

To enable the author to easily write the generator, the WB provides a number of primitives that can be combined for powerful effects including: motion interpolation primitives [RCB98]; warping and time warping [WP95] along with simpler motion editing methods such as splicing, scaling, etc.; and blending and coherent noise from Perlin and Goldberg [PG96].

Coordination between the gesture and the generator is accomplished by symbolic *annotations* on the keyframe curves. Annotations give a name to an artistically meaningful moment in the motion, e.g. `start-anticipation` or `launchTime` and `landTime` for a jump motion. The set of annotations for a gesture is specified by the author, to be whatever they need to capture the artistic intent of their motion. By using symbolic names for the conceptually interesting times, an author can create a new gesture for a given act without changing the generator as long as the motion's structure doesn't change.

These annotations, combined with an author-written procedure for generalizing the motion allows the author to use different methods on different parts of the motion to preserve the artistic qualities that he/she thinks are important. For example, to preserve overlapping timing between two curves in an animation when shortening the overall duration, an author could: proportionally scale the overlapping segment, keep the overlapping segment constant, or proportionally scale the overlapping segment subject to a minimum constant time (e.g. to ensure that the overlap is seen by the user). Any of these properties of the original motion may be important artistic intent. Expressing the generalization process as an author-written procedure enables this intent to be preserved, which more automatic approaches do not. An example of an execution of an act is shown in Figure 8.

The ability to abort an act, which is important to allow arbitrary transitions in some cases, is specified in an additional function for each act. These functions typically allow the act to be aborted unless it is physically impossible. For example a jump act can be aborted at any point except when the character is in the ballistic motion in the air. Encoding

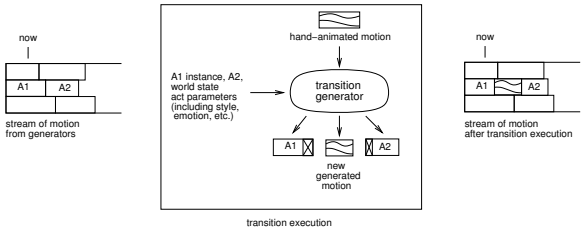

Figure 9: *Transition mechanism.*

this information procedurally allows the author's knowledge of the meaning of the act to inform coherent motion.

Transitions are supported in two ways. In addition to the parameters to the act, each generator is written to accept any initial state of values and tangents for the degrees of freedom that it controls, for instance, by using blending or warping primitives. This is adequate for handling transitions between most acts and transitions between aborted acts and the next act to execute. For some cases, however, a specialized transition is needed. The system supports this by allowing arbitrary *transition generators* to be written by the author. A transition generators takes an executing act and the act to follow, and can remove motion and/or create and insert new motion at the join. This process is illustrated in Figure 9.

Although transition generators are theoretically N^2 in the number of acts, we have found that only a few strategic ones are needed. Most transitions can be effectively handled by the act's generator itself.

7. Usability

Given the complexity of the authoring system we have presented, an important question is how usable is it for authoring interactive characters. In this section we discuss the ease of expressing envisioned interactive characters, brittleness of these characters, and ease of debugging. These observations come from experience training over a half dozen character builders (our term for authors of interactive characters), and from over a dozen interactive characters built by these character builders. Several of these interactive characters are publicly available from www.zoesis.com.

While Gertie is by no means a simple language, we have found authoring in it to be economical and intuitive once a character builder is trained. This is in large part due to the hierarchical nature of interactive minds, and the locality of the Gertie language structures for specifying properties of these minds, and in part because many of the potential complexities of authoring turn out to be sparse in practice. For example, the authoring of `defconflict` expressions in Gertie is theoretically $O(N^2)$ in the number of functions in a character mind, and the number of needed `succeed-when` and `fail-when` expressions are potentially $O(E * S)$ where E is the number of expressions in the Gertie program and S is the number of types of situations to be sensed and reacted to. In practice, both `defconflicts` and `whens` tend to be sparse in character minds. The reason for this is the inherent abstraction and hierarchical nature of minds. An appropriate `succeed-when` or `fail-when` applied to a goal or behavior often obviates the need for similar `whens` in the same behavior because the effects of the success or failure propagate up to parent behaviors and cause children behaviors to be aborted. The result is that `whens` tend to be $O(S)$ with a small constant. Similarly, because the suspensions caused by a `defconflict` effectively block all components of that behavior and behaviors that follow it sequentially, `defconflicts` are also sparse in practice.

In addition, this sparsity of annotations is not an additional complexity of the language that must be explicitly managed by the author. Rather, there are typically natural, local, intuitive positions where a `fail-when`, `succeed-when` or `defconflict` is appropriate to be placed to capture the intended artistic effect.

Debugging in Gertie can be complex. This is in large part because of the complexity of working in a concurrent language. One interesting point however, is that authors typically find debugging to be more onerous when using Gertie as a general-purpose concurrent language, and easier to debug when using it as a model of mind. The organizing structure of concurrent functions as goals being pursued in parallel, conflicting functions as behaviors that conceptually should not be performed by the character at the same time, etc., seems to allow for more intuitive use of the complexity of the language.

One surprising property of interactive characters written in Gertie is on the brittleness/robustness issue. These characters are surprisingly robust. They rarely crash, and bugs in the character behavior usually resolve themselves quickly enough to be judged by the user as part of the personality rather than a bug in the program. This robustness appears to be the result of Gertie's reactive mechanism. The same reactivity that allows the character to adapt to an unpredictable world seems to allow it to adapt and recover from unpredictable internal bugs as well. (Memory bugs are an exception to this. Adding automatic memory management would be a valuable improvement to the language.)

8. Results and Conclusion

We have presented an architecture for the authoring of personality-rich, highly interactive characters. This system is composed of two main components: (1) Gertie, a programming language with unusual language features specifically designed for the expression of procedural artistic knowledge for interactive behavior; and (2) the WB, an action system designed to enable artists to combine the expressive power of hand-crafted keyframe animation with artist written procedures that generalize the motion while preserving the artistic intent.

We have described how this system enables important features of interactive characters to be achieved, and presented examples from an implemented interactive character.

The system is practical, with over a dozen interactive characters implemented to date. Characters built using this technology have been demonstrated at juried conferences including ACM SIGGRAPH 2001 Emerging Technologies, won awards for both entertainment content and technology, and been commercially deployed.

References

[AR02] ANDRE E., RIST T.: Presenting through performing: On the use of multiple lifelike characters in knowledge-based presentation systems. In *Proc. of the Second International Conference on Intelligent User Interfaces* (2002), pp. 1–8.

[BDI*02] BLUMBERG B., DOWNIE M., IVANOV Y., BERLIN M., JOHNSON M. P., TOMLINSON B.: Integrated learning for interactive synthetic characters. In *ACM Trans. on Graphics* (2002).

[BPW93] BADLER N. I., PHILLIPS C. B., WEBBER B. L.: *Simulating Humans - Computer Graphics, Animation, and Control*. Oxford University Press, 1993.

[FTT99] FUNGE J., TU X., TERZOPOULOS D.: Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proceedings of SIGGRAPH 1999* (New York, 1999), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH.

[GM01] GRATCH J., MARSELLA S.: Tears and fears: Modeling emotions and emotional behaviors in synthetic agents. In *Proc. of the 5th International Conference on Autonomous Agents* (2001).

[Gra00] GRASSIA F. S.: *Believable Automatically Synthesized Motion by Knowledge-Enhanced Motion Transformation*. PhD thesis, Computer Science Department, Carnegie Mellon, 2000.

[HP97] HODGINS J. K., POLLARD N. S.: Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 1997* (1997), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH.

[Joh95] JOHNSON M. P.: *Exploiting Quaternions to Support Expressive Interactive Character Motion*. PhD thesis, Massachusetts Institute of Technology, 1995.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *ACM Transactions on Graphics* (2002), vol. 21 of *3*.

[LB93] LOYALL A. B., BATES J.: Real-time control of animated broad agents. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society* (1993).

[Loy97] LOYALL A. B.: *Believable Agents: Building Interactive Personalities*. PhD thesis, Computer Science Department, Carnegie Mellon, 1997.

[LZGB99] LESTER J., ZETTLEMOYER L., GREGOIRE J., BARES W.: Explanatory lifelike avatars: Performing user-centered tasks in 3d learning environments. In *Proc. of the Third International Conference on Autonomous Agents* (1999).

[MS02] MATEAS M., STERN A.: *Architecture, Authorial Idioms and Early Observations of the Interactive Drama Facade*. Tech. Rep. CMU-CS-02-198, Department of Computer Science, Carnegie Mellon University, 2002.

[MTT87] MAGNENAT-THALMANN N., THALMANN D.: The direction of synthetic actors in the film rendez-vous à montréal. *IEEE Computer Graphics and Applications 7*, 12 (1987).

[Nea96] NEAL REILLY W. S.: *Believable Social and Emotional Agents*. PhD thesis, Computer Science Department, Carnegie Mellon, 1996.

[PG96] PERLIN K., GOLDBERG A.: Improv: A system for scripting interactive actors in virtual worlds. *Computer Graphics 30*, Annual Conference Series (1996), 205–216.

[RCB98] ROSE C., COHEN M. F., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5 (1998), 32–40.

[Rey87] REYNOLDS C. W.: Flocks, herds, and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH 1987* (New York, 1987), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 25–34.

[RJ99] RICKEL J., JOHNSON W. L.: Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence 13* (1999), 343–382.

[Sen98] SENGERS P.: *Anti-Boxology: Agent Design in Cultural Context*. PhD thesis, Carnegie Mellon University Department of Computer Science and Program in Literary and Cultural Theory, 1998.

[WK88] WITKIN A., KASS M.: Spacetime constraints. *Computer Graphics 22* (1988), 159–168.

[WP95] WITKIN A., POPOVIĆ Z.: Motion warping. *Computer Graphics 29*, Annual Conference Series (1995), 105–108.