SmartBody: Behavior Realization for Embodied Conversational Agents

Marcus Thiebaux
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA, 90292 USA
thiebaux@isi.edu

Andrew N. Marshall
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA, 90292 USA
amarshal@isi.edu

Stacy Marsella
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA, 90292 USA
marsella@isi.edu

Marcelo Kallmann
University of California, Merced
School of Engineering
5200 N. Lake Road
Merced, CA, 95343 USA
mkallmann@ucmerced.edu

ABSTRACT

Researchers demand much from their embodied conversational agents (ECAs), requiring them to be both life-like, as well as responsive to events in an interactive setting. We find that a flexible combination of animation approaches may be needed to satisfy these needs. In this paper we present SmartBody, an open source modular framework for animating ECAs in real time, based on the notion of hierarchically connected animation controllers. Controllers in SmartBody can employ arbitrary animation algorithms such as keyframe interpolation, motion capture or procedural animation. Controllers can also schedule or combine other controllers. We discuss our architecture in detail, including how we incorporate traditional approaches, and develop the notion of a controller as a reactive module within a generic framework, for realizing modular animation control. To illustrate the versatility of the architecture, we also discuss a range of applications that have used SmartBody successfully.

Keywords

Virtual Humans, Conversational Characters, Character Animation

1. INTRODUCTION

Researchers demand much from their virtual human creations. We want them to be *responsive*; that is, they must respond to the human user as well as other unexpected events in the environment. They must be *believable*; that is, they must provide a sufficient illusion of life-like behavior that the human user will be drawn into the social scenario. Finally, they must be *interpretable*; the user must be able to interpret their response to situations, including their dynamic cognitive and emotional state, using the same verbal and non-verbal behaviors that people use to understand one another. The burden of realizing responsive, believable and interpretable behavior falls in large measure on the animation algorithms and their coordination that constitute the virtual human's "body".

The animation of the body must address a range of technical challenges. To be responsive to unexpected events, a body must be interruptible. Life-like behavior may require multiple parts of the body to be in motion. Gaze, for example, requires coordination of eye movements, head/neck movements, twisting of the joints in the torso as well whole-body stepping movements as necessary. To support a coherent interpretation, body movements may also have to be coordinated and synchronized with each other and with external modules. For example, gestures, head movements, eyebrow lifts, eye flashes and postural shifts augment spoken dialogue in various ways, and their animation must be synchronized in time with each other and with the speech audio.

A variety of approaches have been used to animate virtual humans. Procedural animation readily supports bodies that flexibly point and look at events in the virtual world. However, it is challenging to create procedurally animated bodies that also behave life-like or visually realistic. Conversely, to achieve life-like, highly expressive behavior, researchers have animated the body with pre-defined motion capture or hand-animated sequences of fairly long duration. As a result, the body may not be sufficiently responsive to events, and its outward behavior may not reflect its dynamic cognitive and emotional state in detail.

Our experience has been that a combination of animation approaches may be required to achieve responsive, life-like and interpretable behavior. In previous work, we presented a preliminary motion control architecture to help achieve this combination of approaches [1]. The architecture was based on controllers that could be hierarchically interconnected in real-time in order to achieve continuous motion. This approach has evolved into a general character animation system called SmartBody, an open source modular framework for realizing virtual humans and other embodied characters.

Controllers in SmartBody can employ arbitrary animation algorithms such as keyframe interpolation, motion capture

Cite as: SmartBody: Behavior Realization for Embodied Conversational Agents, Thiebaux, M., Marshall, A., Marsella, S., Kallman, M., *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May,12-16.,2008,Estoril,Portugal,pp. 151-158.

or procedural animation. Controllers can also schedule or blend other controllers. Having composable controllers provides flexibility and a wide range of potential behaviors from a smaller set of simpler components. However, it also raises the issue of how behaviors generated by these controllers combine, synchronize and interact. For example, a proper applied nod must avoid laterally re-orienting the head away from a gaze target. Conversely, the gaze controller must properly realize the vestibulo-ocular reflex while nodding.

In creating SmartBody, we seek to address several goals. Fundamentally, we need a platform that supports research in animation approaches but can also be employed in a range of applications. In addition, we envision SmartBody's open source architecture as a means to facilitate collaboration and as a way for groups to leverage each other's work. For example, one group's gaze controller may conceivably be combined with another group's stepping or walking controller. Further, because the crafting of an ECA (Embodied Conversation Agent) application is such a technologically and artistically daunting task, we hope to lower the barrier of entries by creating a open source system that can be used in a large variety of research efforts and applications. To help facilitate this goal, SmartBody was designed around the SAIBA framework's Behavior Markup Language (BML) standard [2].

In this paper, we present an overview of SmartBody. We describe the core elements of the architecture which include behavior scheduling and synchronization, and the hierarchical organization of motion controllers. We follow up with our results and a discussion of current, related and future work.

2. EXAMPLE

Smartbody implements the behavior realization component of the SAIBA embodied conversational agent framework. That is, it transforms performance descriptions, specified in BML, into character animation and synchronized audio. While each SmartBody process can control multiple interacting characters, we will only discuss a single SmartBody character to simplify the exposition.

Here is an example request, to which we will refer throughout the paper, for the character Alice, addressing Bob:

```
vrAgentBML request Alice Bob SPEECH_24
<?xml version="1.0" encoding="UTF-8"?>
<act>  <bml>
  <speech id="b1">
    <text>
    Wait! That's <tm id="not"> not what I mean.
    </text>
  </speech>
  <sbm:interrupt id="b2" act="SPEECH_23"
                 start="b1:start"  />
  <gaze id="b3" target="Bob" start="b2:start" />
  <head id="b4" type="SHAKE" stroke="b1:not" />
</bml> </act>
```

Table 1: Example BML performance request

Note the BML is preceded by simple routing tokens identifying the type of message, who is speaking, who is addressed, and an act identifier. The BML specifies four behaviors, identified as "b1" through "b4". The primary behavior is Alice's speech. To ensure old speech is completed or cancelled, the sbm:interrupt behavior cancels act "SPEECH_23". Simultaneous to both interrupt and speech, the gaze behavior instructs Alice to look at Bob. Since the gaze does not define end or relax sync-points, the gaze behavior is assumed to persist until the character is directed to gaze elsewhere. And finally, Alice reinforces her contradiction with a simple head shake, synchronized to the word "not". Figure 1 shows these scheduled behaviors on a timeline.

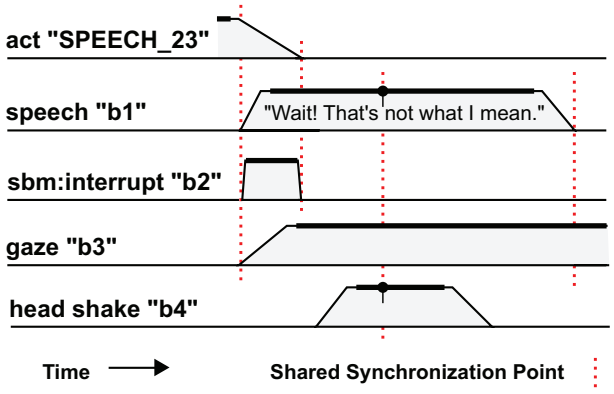

Figure 1: Timeline of the scheduled example behaviors

This example identifies several key issues for SmartBody. First, behavior in SmartBody is realized by the scheduling and blending of multiple controllers. For example, separate controllers are used gaze, rhythmic head movements, and lip visemes while speaking. These controllers are realized by differing techniques. The viseme controller uses weighted morph targets. The gaze uses a procedural controller that can look at and track objects in the virtual world. And the headshake is realized by another, simpler procedural controller. These differing controllers must work together. Specifically, the controllers must be organized in time and evaluation order to realize prescribed transition and blend simultaneous motion. For example, how should the gaze interact with the head shake? If their respective motions are simply added or interpolated, Alice's gaze may not reach the target or the gaze controller may overwrite the neck motions of the nod. The blending of controllers is discussed in Section 4.2.3. Further, the behavior generated by these controllers must be synchronized appropriately. For example, the viseme and gesture animations must be tightly synchronized with the phonemes in the dialog audio and the shake synchronized with the word "not." Failure to achieve such synchronization can significantly impact realism as well as an observer's interpretation of the behavior. Scheduling is discussed in Section 4.1.

3. ARCHITECTURE OVERVIEW

The SmartBody architecture, depicted in Figure 2, works within a network of modules interacting via a shared messaging system. Each SmartBody process can host one or more characters. Inputs include BML commands, speech timing data, and world state updates. Smartbody also outputs character state updates and performance progress feedback (e.g., starts and stops) to other coordinating compo-

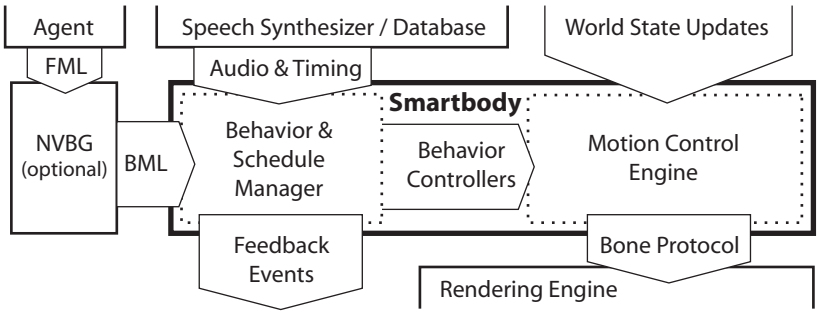

Figure 2: SmartBody processes BML, speech data, and world state to generate animation.

nents over the messaging framework.

The behavior processing components of Smartbody can be broken into two parts: a behavior & schedule manager, which parses the incoming BML, and a motion controller engine, which evaluates the hierarchical schedule of motion controllers responsible for the motions for each character.

In most of our applications, the BML comes from a Non-Verbal Behavior Generator (NVBG) [3] that generates the BML from the agent's communicative intent. The output of the behavior & schedule manager is a hierarchical schedule of motion controllers and events that specifies the motion and synchronization of the characters . The behavior & schedule manager creates and makes necessary edits to this schedule, usually comprised of inserting new motions. Specifically, each BML request is mapped to skeleton-driving motion controllers and feedback events in the schedule.

SmartBody currently supports eight types of BML behaviors: *body* postures, animations, *head* motions like nodding and shaking, *gaze*, *face* for facial action units [4][5] and visemes, *speech*, *sbm:event* for feedback notifications, and *sbm:interrupt* interruption. The behavior & schedule manager maps most behaviors directly to a single motion controller, including animations, postures, head motions, and gaze. However, some BML behaviors invoke network messages or schedule edits. Event behaviors can notify other modules about behavior progress, or possibly facilitate the coordination of multiple character interaction. For example, event behaviors emit a message, scheduled to coincide with a specific point in a behavior, such as when the stroke of a gesture occurs, a gaze acquires a target, or when the dialog is completed. Interrupt behaviors result in schedule edits that modify the blend weights of previously scheduled controllers and visemes during the interrupt.

The schedule & behavior manager must handle both internally and externally generated timing events. For example, speech behaviors schedule audio control commands and lip visemes. The timing of these commands comes from an external audio source, as shown in Figure 2. SmartBody supports both synthesized speech using text-to-speech synthesizers, and pre-recorded audio via a file based audio and timing database. Despite the distributed architecture, latency beteen modules is insignificant. The speech synthesis step introduces an inevitable delay, but the external speech synthesis module implements caching of this data to minimize this.

The second major component is the motion controller engine, which evaluates the hierarchical schedule of motion controllers responsible for the motions for each character. For each frame of animation, the controller engine executes and combines the immediate motion controllers to generate a set of skeletal joint rotations and translations.

The results are sent over a fast network protocol to the system's rendering engine. The renderer applies the skeletal motion to the character's deformable mesh, displayed within the application scenario. This separation from the application rendering system keeps SmartBody portable to multiple rendering engines.

The separation of animation calculation from the rendering environment also allows SmartBody to work in a coarsely distributed architecture. Multiple SmartBody processes can each manage one or more characters, sending all skeleton updates to a shared rendering engine. As a platform for experimental character animation, this facilitates the use of computationally heavy controllers for behaviors like physical balance. At the moment, characters hosted on separate processes cannot interact with each other. We intend to solve this problem with the integration of a world state protocol capable of subscription-based tracking of objects, including the position and orientation of individual character joints or environmental objects outside of SmartBody's control.

4. TECHNICAL DESCRIPTION

The following details the two major components of SmartBody.

4.1 Behavior & Schedule Manager

The Behavior and Schedule Manager parses BML into a set of behavior requests and time references. The time references refer to the standard BML synchronization points: start, ready, stroke, etc. The behavior requests and time references are used to realize a schedule of motion controllers that animate the characters.

A key concern for the scheduling is the range of behaviors that need to tightly synchronized with speech, including not only behaviors such as visemes that represent the physical manifestation of the speech but also a range of nonverbal behaviors such as gestures, head movements and eye brow movements that can be used to augment, emphasize or replace aspects of the spoken dialog. Such behaviors often rely on audio with fixed timing and non-standard sync-points at word breaks (in addition to the core BML synch-points). When parsing speech behavior, SmartBody generates the behavior request and time references normally, but also submits a request for speech timing and audio playback/stop commands. This speech audio and word break timing comes from either a speech synthesis system or an audio database.

After speech and all other BML behaviors are parsed, the behavior scheduler pauses processing until receiving the speech timing reply message.

Next, the behavior scheduler assigns absolute timing to the time references of each behavior request. The behaviors are processed in BML order, such that the first behavior has no timing constraints and is assigned the default timing for the behavior type. Subsequent behavior time references are assigned by scaling the behavior's default sync-point timing to any constrained durations. Partially constrained time spans, where start or end sync-points may not be defined, are time-scaled to match the adjacent span of the behavior to avoid any discontinuities in animation speed. Because some time constraints can result in scheduling behaviors into the past, a final normalization pass may shift all scheduled time markers values forward in time. During this pass, the earliest behavior start and latest behavior end are noted and later used to schedule upstream feedback events about the performance progress.

When inserting new controller tracks, the schedule manager must take into account the evaluation order of controllers that affect common joints. As we will explain further in subsection 4.2.3, where we describe blending techniques, ordering affects some types of simultaneous motion primitives. The behavior & schedule manager groups behavior controllers into categories of behavior types. The categories are ordered such that lower categories apply motion to the animation channels first, allowing modifications and overwrites by higher controller categories. Poses and motion textures fall into the lowest category, providing a base pose for the character skeleton. Higher categories include full body animations, spine and gaze movement, head and neck movement, and facial expressions, in that order. Generally, higher categories implement more refined motions.

In addition to controllers, the schedule & behavior manager may attach events to the time references in the form of broadcasted messages. This automatically occurs for the feedback events of the performance request, signaling the completion of the performance. Additionally, several behaviors produce their own messages, such as the audio commands of the speech behavior, and the *sbm:event* message which broadcasts a string in sync with the behavior stroke. In the future, we intend to support synchronizing to remote events via the *wait* behavior, thus enabling a higher level of coordinated performances between characters and with the environment.

4.2 Motion Control Engine

The motion controller engine provides a framework in which scheduled controllers can generate and apply skeletal joint transformations. The engine maintains the articulated skeletal model, and a compact buffer of channels representing the animated joint parameters. A motion controller is a self-contained module that affects the motion of joints in a skeleton or some subset, which can be combined with other concurrent motion primitives to build up more complex behaviors.

Within the engine, controllers are organized in a schedule, not only as a sequence of operations with overlapping transitions, but also as a prioritized layering of concurrent tracks that may affect common joints. This concurrency allows us to compose simultaneous behavioral primitives with ease. Looking at our example in Table 1 and Figure 1, the primary body motion primitive is a Gaze controller, which performs throughout the duration of the exchange. At a key synchronization point, before Alice speaks the word "not", a HeadShake controller applies its motion to affect a subset of the joints otherwise controlled by the Gaze action.

When a controller executes, it is supplied with a reference to a writeable channel buffer for storing the values for joints on which it is configured to operate, shown in Figure 3. This buffer initially contains the output channels from the controller lower in the immediate controller stack, which has just been executed. We call the contents of this buffer a partial-calculation of the motion for the current frame. The current controller can read this buffer as input to integrate into its calculation. The channel buffer thus serves to pass along the partial results of motion calculations, by accumulating the motion effects produced by the ordered stack of concurrent controllers.

Some controllers also require read-access to the skeletal joint parameters from the previously completed animation frame. The ability to reference the previous frame is important for calculating spatial relationships of completed motions in the world, such as to maintain velocity control or other spatial continuity as it executes. The controller framework supplies the controller with a reference to the skeletal model, to directly query the joint hierarchy for its local and global matrices.

A controller can also maintain and effectively 'drive' its own *subordinate* controller, by passing along the channel buffer as input. Figure 4. The results are then retrieved by the super-ordinate, and used as a secondary input to its own motion. Controllers that accept a single subordinate (see the Blend discussed below), can be chained together. A controller with multiple subordinates is a branching controller.

At the top level, our controller engine operates on a simple stack of controllers that are successively combined. However, the use of branching controllers allows for a general tree-like arrangement of modules, and the aggregation of controllers as a self-contained unit within a larger schedule. This model will become useful for the creation of more advanced controllers such as for walking and running, which must maintain a more complex internal state to ensure continuity of their underlying scheduled motion cycles.

4.2.1 Motion Controller Types

For convenience, we can distinguish between ordinary motion controllers and meta-controllers which execute subordinate controllers. Motion controllers simply generate motion and apply it to the channel buffer one way or another.

Pose: The Pose controller is the most basic, writing prestored values into the joint channels for full or partial-body configurations. Typically, a character's schedule is assigned an underlying full-body pose of indefinite duration, to which it returns when there are no overriding motions.

KeyFrame: The KeyFrame controller extends the idea of the pose controller with a time-stamped series of pre-defined poses, which are then interpolated on demand.

HeadTilt/Nod/Shake: These neck joint controllers add their joint adjustments to the underlying pose or motion by first reading the channel values, and then modifying their relevant angular components, as described in subsection 4.2.3.

StepTurn: The StepTurn controller plays pre-defined stepping-turn animations of the lower body, and modifies

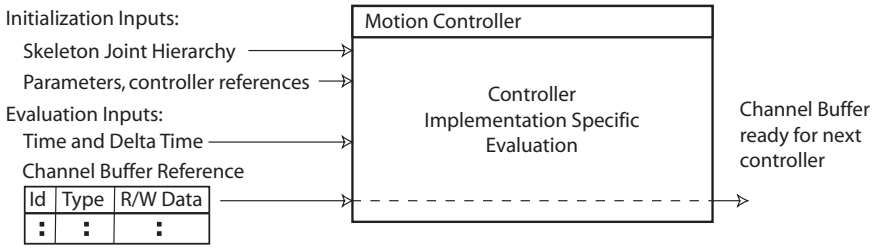

Figure 3: A controller module is initialized with configuration parameters, a skeleton reference, and optional references to subordinate controllers. At run-time, it is supplied with clock time and delta, and a read-write buffer of channels on which it acts.

the resulting joint values in order to acheive a desired turn angle. This is sufficient for clearly defined styles of body turns that take place on one spot, such as in conjunction with gazing at a moving target that moves behind the head.

Gaze: Our Gaze controller coordinates the behavior of joints in the back, neck, head, and eyes in order to manipulate a character's *line of regard*, and general communicative appearance of visual or frontal attention. Each joint is driven by a mini-controller routine that fulfills a designated portion of the alignment task, yet operates independently (within its limits) to maintain tracking even while parent joints may be influenced by other controllers. In the eyes, this mimics the vestibulo-ocular reflex, but the same principal applies to the tracking contribution of the head and neck. The Gaze implementation also addresses more subtle issues regarding gaze aversion, the apparent intensity of a gazing action, deictic head gestures, and dominant/submissive spinal posture. The API defines a set of major control keys (BACK, CHEST, NECK, HEAD, and EYES) that are internally mapped to the actual spine and head joints. This abstraction provides a standardized set of handles for tuning of parameterized behavior, distinct from the specific joints used in the skeleton for a particular character.

4.2.2 Meta-Controller Types

Meta-controllers manipulate the behavior of subordinate controllers by intercepting their run-time input parameters and output channels.

TimeShiftWarp: The TimeShiftWarp controller helps coordinate the timing and scheduling of subordinate motions to fit desired behavioral events. Clock time and delta are mapped to new values, and passed to its subordinate controller. This allows varied start times and playback speeds of any specified motion.

Blend: The Blend controller, illustrated in Figure 4, combines its input channel buffer with the output buffer of a subordinate controller, using an adjustable spline-based blending curve.

Schedule: The Schedule controller maintains a stack of subordinate controllers and enables their scheduling as tracks of motion applied over time. Each track may use the Blend and TimeShiftWarp controllers to adjust blending weights and execution speeds. Since the Schedule is itself a controller, a new Schedule can be dynamically invoked to serve as another subordinate controller, effectively encapsulating a group of related scheduled motions as a single unit.

4.2.3 Simultaneous Controller Interaction

In our previous work [1] we attempted to make a clear distinction between source and connector controllers, as a convenient way to frame a motion combinatorial system. Connectors implemented the blending, scheduling, or other filtering of motion sources required for combining motions, without defining an intrinsic motion of their own. In our experiments, we find it convenient to view a motion controller as not just a passive source of motion, but an active module that may determine the manner in which it is combined.

For example, the gaze controller allows for non-uniform blending weights to be applied to a character's joints. Because Gaze controls many joints in the spine and neck, it would override any underlying posture or body motion. Non-uniform weights allow some of the underlying posture to influence the lower range of the spine as desired, while maintaining full control of the neck and eyes. Consider a seated character, where gaze action on the back and chest is constrained by the chair. Attempting to implement this with a generic blend controller would be unwieldy.

The ordering of controllers within the stack of concurrent tracks determines the precedence of one controller's effects over another, as they are successively executed. However, this ordering by itself does not address the complexities that can arise when different controllers interact over the same joints. In order to coordinate actions, the second controller cannot always be blended with the first using simple linear interpolation of the raw channel quantities.

In our Alice example, the Gaze controller takes precedence in coordinating the rotations of spine, head and eye joints, such that each joint contributes to the overall task of obtaining alignment of Alice's line of sight with her target, Bob. The Gaze controller handles this task in part by having each joint, such as an eye, track toward the target independently of its parent joint, within its prescribed range of motion. We wish to maintain the important target tracking contribution of the neck while Alice shakes her head in disagreement.

Our HeadShake controller defines a smooth motion curve which oscillates the neck joints about their vertical axis. Rather than interpolating between motion sources, the HeadShake *adds* its motion directly to the output of the Gaze controller, so that if Alice's head is turned to the side to face Bob, it remains turned, and the motion is applied relative to this underlying angle. This result realizes the expected behavior of a composable HeadNod or HeadShake controller in a simple manner.

Our controllers can utilize information about the context in which they are running, in order to adapt their behavior appropriately. This context can include the partial-calculation, the previous frame's skeleton, or even output from a designated subordinate controller.

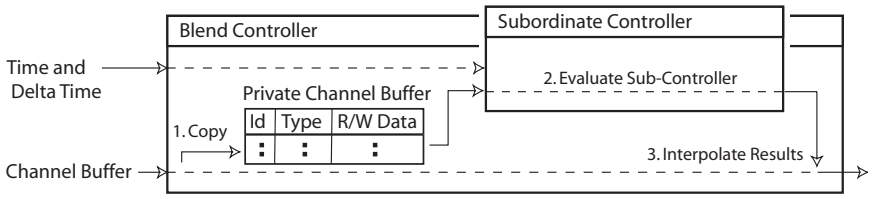

Figure 4: Blend interpolates between the output of a controller lower in the stack with the output of a subordinate controller.

5. SYSTEM INTEGRATION

A key goal for SmartBody is to be applicable to different systems with different agent capabilities, different types of characters and different renderers. To help realize wide applicability, we have had to address several key concerns. First, there must be an art pipeline, a flexible way for art assets to be incorporated into SmartBody by character creators and animators. Second, there must be some standardized API to control SmartBody that is amenable to use by different research groups and application builders. Finally, there must be a straightforward way to hook up SmartBody to differing rendering/game engines. In this section, we briefly touch on these input/output aspects.

Art Pipeline: Although research platforms are often built with a single character, we expect SmartBody might be used across both research systems and applications with perhaps a wide range of different characters, different skeletons as well as different art pipelines for creating those characters. Currently, we use Maya to create characters, exporting skeletal animation and geometry information into our simple text data formats, via several Mel scripts and SmartBody-specific tools. Animations are created using traditional key frame techniques, or through motion capture, for use by controllers such as Pose and KeyFrame. We also export the skeleton and character mesh from Maya to the rendering engine using the rendering engine's toolset.

SmartBody can work with other 3D packages by writing export scripts that write animation and skeleton data into SmartBody's simple text format. We also plan to support a more generic Collada [6] bassed asset pipeline.

BML Input: As discussed above, SmartBody takes the SAIBA framework's BML commands for the behavior control. We are largely noncommittal as to how the BML is crafted. In some SmartBody applications (see section 6) use the BML is handcrafted. In others, the BML is generated automatically (e.g., by the virtual human's "brain"). Most of our current applications using SmartBody generate BML using NVBG [3], a rule-based module that analyzes the virtual human's communicative intent, emotional state, and speech text to determine various nonverbal behaviors including facial expressions and gestures.

Rendering: Finally, there must be a straightforward way to connect SmartBody to different rendering engines. Our current integration with rendering systems is via a network interface. We have full and partial integrations with the following renderers: Unreal 2.5 (full), Gamebryo, Ogre3D and Half-Life 2 (via a user mod).

6. RESULTS & APPLICATIONS

Two key goals for the SmartBody project were generality and ease of use. We wanted an animation system that could be used by others in a wide range of systems. Further, we wanted to simplify and speed up the process of crafting virtual human and ECA applications. All of these goals have been demonstrated by Smartbody's use in seven different applications, including fielded training applications, commercial systems and research systems. Here we briefly discuss four, SASO-ST[7], ELECT[8], Virtual Patient[9], and Virtual Rapport[10].

In the Stability and Support Operations Simulation and Training (SASO-ST) project (See Figure 5), a single Smartbody controls two characters in a 3-party conversation with a human participant. SASO-ST is a platform for virtual human research. ELECT, on the other hand, is a fielded training system designed to teach cross-cultural negotiation skills. In ELECT, the SmartBody was used by an outside game company. The company crafted their own characters and added animations to SmartBody's existing repertoire. The Virtual Patient system uses SmartBody to realize a virtual boy with a conduct disorder, as a demonstration of how virtual humans can be used as standardized patients for training health care professionals. In all these systems, NVBG is used to generate the BML annotations from the speech text. A few behavior rules were changed for the different applications, but most of the rules were common across all projects. Across all the various systems that have used SmartBody, there has been very significant reuse of keyframe animation assets, and they all use the same procedural controllers.

However, the surrounding system in which SmartBody is being used has differed considerably. In the SASO-ST application, voices are generated using text-to-speech, in real time with timing information from the Rhetorical system. For ELECT and the Virtual Patient, voices are pre-recorded. Also, each of these systems use entirely different agent designs to drive SmartBody. SASO-ST uses the Austin virtual human architecture that incorporates task reasoning, emotion modeling and dialog management, while ELECT and Virtual Patient use lighter weight reactive agents that map the users input dialog to the agent's response. On the other hand, the Virtual Rapport agent does not interact with dialog at all. Rather it tries to provide nonverbal listening feedback such as posture shifts and head nods to a user speaking to the system. It determines its responses using vision and audio data derived from the user's head movements and speech.

In summary, different groups both inside and outside our organization created these systems and they used different sensor, agent and speech technologies to do it. Nevertheless, SmartBody has proven to be efficient and flexible, greatly reducing the time and cost to build the virtual humans in the respective projects.

Figure 5: SASO-ST, ELECT & Virtual Patient.

7. RELATED RESEARCH

A wide range of computer animation techniques are available for the animation of interactive virtual humans, for instance: walking [11, 12], reaching and object manipulation [13, 14, 15], inverse kinematics [16, 17], parameterized models [18] and keyframe interpolation of designed or motion captured [19] keyframes. These works produce compelling results, however they do not address synchronization issues when different methods must be combined to achieve a composed, full-body behavior.

Instead of focusing on individual animation techniques, SmartBody is based on the concept of achieving complex behaviors by the composition of motion generators, with an emphasis on simple primitives. Such an approach is supported by strong evidence found in neuroscience research dealing with biological motion organization [20, 21].

Several previous works on computer animation [22, 23, 24] have addressed composition of motion controllers, yet are mainly limited to achieving smooth transitions between controllers with blending. For instance, Boulic et al [24] propose a system where arbitrary sets of joints can be activated and blended with smooth transitions, but without handling scheduling or hierarchical motion flow. Animation packages available in recent game engines also offer similar capabilities. For instance the Ogre3D game engine uses the notion of *animations* for key frame animations and *controllers* for user-implementable motion generators; however no framework is given for solving the scheduling and hierarchical needs addressed by our approach.

The problem of automatically composing motion controllers in order to achieve a desired behavior is more difficult to address in general situations. An approach based on selecting individual controllers for guiding a simulated character under dynamics has been proposed by Faloutsos et al [25], however it does not address our scheduling and hierarchical needs.

The specific problem of synthesizing realistic behaviors for conversational characters has been also addressed by several researchers [26, 27, 28, 29]. In general, for conversational characters, the usual approach has been to compose annotated sequences in order to match timing constraints.

For instance, the BEAT system [27] correctly addresses time-synchronized sequential animations driven by the speech text. However its scheduler treats non-verbal behaviors as immutable, unable to adapt them to scheduling constraints, nor refer to their inner structure. As such, preparatory action and transitional behaviors must be independently scheduled, or left to the animation engine to coordinate. In contrast, Smartbody's scheduler generalizes timing constraints beyond spoken word boundaries and correctly handles non-verbal behavior. The scheduler understands notions of behavior transition corresponding to BML's ready and relax sync-points, as well as important moments of action such as strokes. That said, BEAT does address the issue of conflicting behavior requests, currently ignored by SmartBody's scheduler.

The Max system [26] also uses a similar notion of primitive controllers, called local-motor-programs, that operate on separate parts of the character: hand, wrist, arm, neck, eyes, etc. Motor programs are then recombined by an abstract motor control program (MCP). Although joint sharing and blending are addressed they do not model generic and flexible reconnections and processing of arbitrary primitive motion controllers.

In a different approach, Stone et al [29] focuses on reusing motion captured sequences, reducing the amount of time in designing and annotating timing constraints, but requiring suitable motion capture performances as input to be reused in the lines to be simulated.

In general, our system differs from these models mainly by providing a unique architecture for the dynamic and hierarchical composition of controllers. In particular, SmartBody offers mechanisms for precise time scheduling, automatic handling of warping and blending operations to meet time constraints, and integration of custom controllers for achiving a wide range of effects. Furthermore, our behavior & schedule manager is able to assemble and orchestrate controllers following input from BML specifications. The capability of hierarchically organizing the motion flow between controllers, both concurrently and sequentially, builds a flexible framework for expressive interactive virtual humans.

8. STATUS & FUTURE

The work on SmartBody has been designed to address several goals. We sought a platform that both lowered the barrier of entry in creating ECAs and was flexible enough for a wide range of applications. The range of applications and research efforts that now use SmartBody provide solid evidence of our progress in achieving that goal. We also sought an open platform that supports research in addressing the challenges of animating ECAs. To help achieve that goal, we are in the process of preparing SmartBody for release to the public. We hope to coordinate with other institutions to develop new controllers and support new applications.

We are also actively expanding SmartBody on several fronts. At the scheduling level, we are expanding our support for BML sync-point notations and core behaviors. At the controller level, we are exploring parameterized stepping and an intelligent step-planning controller. We have recently completed bone-based facial control and are currently test-

ing it. We are also exploring the integration of controllers that employ physical simulations.

You can follow our progress at:
http://www.smartbody-anim.org/

9. ACKNOWLEDGEMENTS

This work was sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM), and the content does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

10. REFERENCES

[1] Kallmann, M., Marsella, S.: Hierarchical motion controllers for real-time autonomous virtual humans. In: Proc. of Intelligent Virtual Agents (IVA'05), Kos, Greece (September 12-14 2005) 243–265

[2] Kopp, S., Krenn, B., Marsella, S., Marshall, A., Pelachaud, C., Pirker, H., Thorisson, K., Vilhjalmsson, H.: Towards a common framework for multimodal generation: The behavior markup language. In: Proc. of Intelligent Virtual Agents (IVA'06). (2006) 105–111

[3] Lee, J., Marsella, S.: Nonverbal behavior generator for embodied conversational agents. In: Proc. of Intelligent Virtual Agents (IVA'06), Marina del Rey, CA (August 21-23 2006) 243–255

[4] P. Ekman, W.V.F.: InvestigatorÕs guide to the Facial Action Coding System. Consulting Psychologist Press (1978)

[5] Ekman, P.: Emotion in the human face. Cambridge University Press (1982)

[6] Arnaud, R., Barnes, M.: COLLADA: Sailing the Gulf of 3D Digital Content Creation. A K Peters, Ltd. (2006)

[7] Swartout, W., Gratch, J., Hill, R., Hovy, E., Marsella, S., Rickel, J., Traum, D.: Toward virtual humans. AI Magazine **27**(1) (2006)

[8] Hill, R., Belanich, Core, M., Lane, Dixon, Forbell, E., Kim, J., Hart: Pedagogically structured game-based training: Development of the elect bilat simulation. In: Poster presentation at the 25th Army Science Conference, Orlando, FL. (Nov 2006)

[9] Kenny, P., Parsons, T.D., Gratch, J., Rizzo, A.: Virtual patients for clinical therapist skills training (2007)

[10] Gratch, J., Okhmatovskaia, A., Lamothe, F., Marsella, S., Morales, M., van der Werf, R.J., Morency, L.P.: Virtual rapport. In: Proc. of Intelligent Virtual Agents (IVA'06), Marina del Rey, CA (August 21-23 2006)

[11] Boulic, R., Magnenat-Thalmann, N., Thalmann, D.: A global human walking model with real-time kinematic personification. The Visual Computer **6**(6) (1990)

[12] Park, S.I., Shin, H.J., Shin, S.Y.: On-line locomotion generation based on motion blending. In: Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA), New York, NY, USA, ACM Press (2002) 105–111

[13] Kallmann, M.: Scalable solutions for interactive virtual humans that can manipulate objects. In: Artificial Intelligence and Interactive Digital Entertainment (AIIDE), Marina del Rey, CA (June 1-3 2005)

[14] Liu, Y., Badler, N.I.: Real-time reach planning for animated characters using hardware acceleration. In: Proc. of Computer Animation and Social Agents (CASA'03). (2003) 86–93

[15] Kuffner, J.J., Latombe, J.C.: Interactive manipulation planning for animated characters. In: Proc. of Pacific Graphics'00, Hong Kong (Oct 2000) poster paper.

[16] Baerlocher, P.: Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures. PhD thesis, Swiss Federal Institute of Technology, EPFL (2001) Thesis number 2383.

[17] Tolani, D., Badler, N.: Real-time inverse kinematics of the human arm. Presence **5**(4) (1996) 393–401

[18] Gu, E., Badler, N.: Visual attention and eye gaze during multipartite conversations with distractions. In: Proc. of Intelligent Virtual Agents (IVA'06), Marina del Rey, CA (2006)

[19] Bodenheimer, B., Rose, C., Rosenthal, S., Pella, J.: The process of motion capture: Dealing with the data. In Thalmann, D., van de Panne, M., eds.: Computer Animation and Simulation '97, Springer NY (Sept 1997) 3–18 Eurographics Animation Workshop.

[20] Thoroughman, K.A., Shadmehr, R.: Learning of action through combination of motor primitives. Nature **407**(6805) (2000) 742–747

[21] Giszter, S.F., Mussa-Ivaldi, F.A., Bizzi, E.: Convergent force fields organized in the frogŠs spinal cord. Journal of Neuroscience **13**(2) (1993) 467–491

[22] Granieri, J.P., Crabtree, J., Badler, N.I.: Production and playback of human figure motion for visual simulation. ACM Transactions on Modeling and Computer Simulation **5**(3) (1995) 222–241

[23] Perlin, K., Goldberg, A.: Improv: A system for scripting interactive actors in virtual worlds. In: Proc. of SIGGRAPH 96, New Orleans, LA (1996) 205–216

[24] Boulic, R., Bécheiraz, P., Emering, L., Thalmann, D.: Integration of motion control techniques for virtual human and avatar real-time animation. In: Proc. of Virtual Reality Software and Technology (VRST'97), Switzerland (Sept 1997) 111–118

[25] Faloutsos, P., van de Panne, M., Terzopoulos, D.: Composable controllers for physics-based character animation. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (2001) 251–260

[26] Kopp, S., Wachsmuth, I.: Synthesizing multimodal utterances for conversational agents. Computer Animation and Virtual Worlds **15**(1) (2004) 39–52

[27] Cassell, J., Vilhjálmsson, H.H., Bickmore, T.W.: Beat: the behavior expression animation toolkit. In: Proceedings of SIGGRAPH. (2001) 477–486

[28] Carolis, B.D., Pelachaud, C., Poggi, I., de Rosis, F.: Behavior planning for a reflexive agent. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle (Sept 2001)

[29] Stone, M., DeCarlo, D., Oh, I., Rodriguez, C., Stere, A., Lees, A., Bregler, C.: Speaking with hands: creating animated conversational characters from recordings of human performance. ACM Transactions on Graphics (SIGGRAPH'04) **23**(3) (2004) 506–513