J Multimodal User Interfaces (2010) 3: 271–284
DOI 10.1007/s12193-010-0051-3

ORIGINAL PAPER

Elckerlyc

A BML Realizer for continuous, multimodal interaction with a Virtual Human

Herwin van Welbergen · Dennis Reidsma · Zsófia M. Ruttkay · Job Zwiers

Received: 20 November 2009 / Accepted: 24 August 2010 / Published online: 17 September 2010

Abstract "Elckerlyc" is a BML Realizer for generating multimodal verbal and nonverbal behavior for Virtual Humans (VHs). The main characteristics of Elckerlyc are that (1) it is designed specifically for *continuous interaction* with tight temporal coordination between the behavior of a VH and its interaction partner; (2) it provides a mix between the *precise temporal and spatial control* offered by procedural animation and the *physical realism* of physical simulation; and (3) it is designed to be *highly modular and extensible*, implementing the architecture proposed in SAIBA.

Keywords BML Realizer · Virtual Humans · Embodied conversational agents · Multimodal behavior generation · Physical simulation · Procedural animation · Coordinated interaction · Mixed dynamics

1 Introduction

As Virtual Humans (VHs) are finding their way into a broad range of applications such as training and tutoring, virtual worlds, (serious) games and various genres of entertainment, interest in flexible behavior markup languages and realizers for the specification and real-time generation of high-quality and subtle multimodal behavior has grown.

This paper presents "Elckerlyc", a BML Realizer for generating multimodal verbal and nonverbal behavior for VHs.[1] A BML Realizer takes a specification of the intended behavior (speech, gaze, gestures, etc.) of a VH—written in the Behavior Markup Language (BML) [9]—and executes this behavior through the VH. Elckerlyc builds upon several earlier projects with VHs that were carried out at the Human Media Interaction lab. During those projects, the need for a number of specific novel characteristics became clear, that were not available in the animation engines that we used. Using the experience gained in the earlier projects, we developed Elckerlyc from the ground up as a state of the art BML Realizer. Four applications have been the most instrumental in this respect: the virtual presenter, the interactive dancer, the reactive virtual trainer, and the virtual orchestra conductor, all described in detail elsewhere [12]. We also built upon our earlier work with behavior specification—most notably MultiModalSync [20] and Gestyle [13].

1.1 Continuous interaction

Elckerlyc implements novel techniques to support animation of *real-time continuous interaction*. The design of VHs often focuses on the combination of speech with gestures in conversational settings. They tend to be developed using a turn-based interaction paradigm in which the user and the system

This research has been supported by the GATE project, funded by the Dutch Organization for Scientific Research (NWO) and the Dutch ICT Research and Innovation Authority (ICT Regie).

H. van Welbergen · D. Reidsma (✉) · Zs.M. Ruttkay · J. Zwiers
Human Media Interaction, University of Twente, PO Box 217, 7500 AE, Enschede, The Netherlands
e-mail: dennisr@ewi.utwente.nl

H. van Welbergen
e-mail: welberge@ewi.utwente.nl

Zs.M. Ruttkay
e-mail: zsofi@ewi.utwente.nl

J. Zwiers
e-mail: zwiers@ewi.utwente.nl

[1] "Elckerlyc" is the protagonist of a Dutch morality play with the same name, written at the end of the Middle Ages. The name translates as "Everyman"; the protagonist represents every person, as they make the journey towards the end of their life.

take turns to talk. If the interaction capabilities of VHs are to become more human-like and VHs are to function in social settings, their design should shift from this turn-based paradigm to one of *continuous interaction* in which all partners perceive each other, express themselves, and coordinate their behavior to each other, continually and in parallel [12, 17]. This requires the realizer to be capable of immediate adaptation—in content and in timing—to the dynamics of the environment and the user. Especially the responsive adaptation of the *timing* of behavior that is being realized requires specific capabilities in Elckerlyc.

1.2 Naturalness and control

Elckerlyc makes use of motion captured motion, procedural animation[2] and physical simulation to steer the movement of the VH. Physical simulation provides physically realistic motion and (physical) interaction with the environment. In van Welbergen et al. [21] we argue that physical realism is one of the aspects of natural motion. Physical controllers can robustly retain or achieve desired movement states (joint rotation, center of mass position, etc.) under the influence of external perturbation. This robustness comes with a disadvantage: precise timing and limb positioning using physical controllers is an open problem [21]. Procedural animation offers precise timing and limb positioning and can make use of many motion parameters. However, it is hard to incorporate movement details such as those found in recorded motion into the mathematical formulas that steer procedural animation. Furthermore, to maintain physical realism, it has to be explicitly authored in the procedural model for all possible parameter instances. Motion (capture) editing techniques retain the naturalness and detail of recorded motion. However, these techniques produce natural motion only when the modifications are small, and the amount of required recordings grows exponentially with the number of motion parameters used. We refer the reader to van Welbergen et al. [21] for a thorough discussion on the naturalness and control offered by different animation techniques.

Elckerlyc offers a mix between, on the one hand, the *precise temporal and spatial control* offered by kinematic animation techniques such as motion capture, keyframe animation and procedural animation, and, on the other hand, the physical realism of physical simulation. Elckerlyc can steer a VH using kinematic animation and physical simulation. These two paradigms can be used in parallel on different body parts, and the assignment of body parts to one of the paradigms can be modified on the fly. We model the force transference from body parts that are kinematically animated to the physically simulated part of the body, increasing the perceived physical realism and physical coherence of the resulting motion. The possibility to specify animation both kinematically and physically also allows one to select the paradigm that requires the least authoring effort for each required animation. Physical interaction with the environment, for instance balancing, are hard to author procedurally, but relatively easy to author using physical controllers in a physically simulated environment. Gestures, on the other hand, need to adhere to strict timing and spatial constraints and typically make use of many control parameters. They are therefore better defined procedurally [21].

[2]We adapt the animation classification of van Welbergen et al. [21], procedural animation always refers to kinematic procedural motion in this paper.

1.3 Abstraction, modularity and extensibility

VH platforms such as Elckerlyc are used by a multi-disciplinary research community—consisting of computer scientists, (computational) linguists, psychologists, and others—interested in studying multimodal human behavior in human–human and human-machine interaction. Using quick prototyping of gestures, facial expressions and body movements, they build VHs that display human-like behavior. This multi-disciplinary community has a very wide range of requirements with respect to the *level of abstraction* provided by, e.g., Elckerlyc. Some users need access to its functionality only in terms of the *possible behaviors* that the VH can display, abstracting away from the details of underlying animations. Others need to exercise detailed control over the *exact form* of the behaviors that they want to study.

To achieve such a separation of concerns, the authors have at a very early stage joined the SAIBA (Situation, Agent, Intention, Behavior, Animation) initiative, and in particular the development of the emerging Behavior Markup Language standard (BML) [9, 23]. The SAIBA initiative provides, among other things, a view on the architectural issues of building a fully functional VH with different layers of abstraction. Within this context, BML is a markup language that allows one to specify the different behaviors that a VH should execute (such as speech, gestures, poses, and gaze), together with their synchronization. Elckerlyc has been designed as a *highly modular and extensible* system that provides, at the same time, high level access to abstract behaviors through the use of BML, and, for those who need it, low level access to the exact execution, scheduling and timing of the behaviors.

1.4 Example application

Below we describe our virtual conductor [14]. This example will be referred to throughout the paper when explaining aspects of the Elckerlyc BML realizer. Although it was originally built when Elckerlyc had not yet been developed, the core elements of the virtual conductor have found their way into Elckerlyc. Video material showing elements described

in the running example can be found on the Elckerlyc showcase.[3]

Running Example 1 (The virtual conductor)
We have built the interactive virtual conductor, a VH that can interactively conduct an ensemble of human musicians. It chooses, schedules, and performs the right conducting movements for a given piece of music. The right hand is almost always indicating the beat; the left hand is often loosely hanging down, but is also used to make additional gestures such as entrance cues. While conducting, audio processing is used to perceive the music being performed. When the musicians go too fast or too slow, the conductor will change the timing of its planned beat gestures to lead them back to the right tempo. The conductor can add gestures on the fly while playing (e.g. "play louder" when the music is too soft), or stop the piece when the music is not good enough.

Of course, any VH needs to be able to plan multi-modal behavior, and to extend or change the planned behavior based on its perceptions. In conversations, people also subtly adapt their timing to each other [12]. Currently, we are developing a research application that will be very close to the virtual conductor with respect to these timing changes: the reactive virtual trainer. The virtual (fitness) trainer will perform an exercise together with a human user in a certain tempo. Since the user will not necessarily perform the exercise with robotic precision, the virtual trainer needs to be able to adapt the timing of its behavior (the movements of the exercise as well as accompanying explanations and motivational utterances) to the timing with which the user performs the exercise.

2 Related work

2.1 Continuous interaction

Continuous interaction needs flexible planning and behavior synchronization and anticipation, not only to internal modalities but also to the environment and participants in the interaction. Thórisson [17] describes an interactive cartoon character engaging in an information exchange with the user. In their system, perception and behavior generation are parallel and ongoing all the time, and behavior plans can be modified last-moment in response to new perceptions and decisions.

Loyall et al. [10] describe a system that allows the authoring of highly interactive motion and demonstrate their approach in an interactive game with a personality-rich character. The behavior of this character is even more tightly coupled to changes in the environment (a bouncing ball, moving mouse, etc.), also with respect to the exact timing. Continuous changes and the unpredictability of this environment require flexible animation planning and execution processes. This is achieved by animating their character using a flexible mechanisms that can interrupt keyframe animation segments and fluently concatenates them.

Like Loyall et al. [10] we offer synchronization to the *anticipation* of user behavior (e.g. what is the tempo that the musicians are playing in). This is not just relevant for games or for a virtual conductor, but also necessary for general conversational capabilities of a VH [12, 17]. We implement more complex behavior realization than Loyall et al. [10], adding a modality for speech and physically specified animation and allowing internal synchronization between modalities.

2.2 BML realization

The Behavior Markup Language (BML) provides a general, Realizer-independent description of multimodal behavior that can be used to control a virtual human. BML expressions (see Fig. 1 for a short example) describe the occurrence of certain types of behavior (facial expressions, gestures, speech, and other types) as well as the relative timing of the involved actions [9].

Elckerlyc is a new BML realizer. Recently several other BML realizers have been developed [1, 4, 16], and existing frameworks for multimodal behavior generation are being modified to support BML [3, 8]. These realizers are used either in turn-based interaction applications [3, 8, 16] or offline, for instance in the reproduction of annotated behavior of real humans [4]. Since these realizers are not specifically designed for continuous interaction they do not support re-timing and re-planning of behaviors that are already in the play queue. Animation is specified by parameterizable keyframes and end effector movement trajectories [3, 4], procedural controllers that support emotional parameterization [3], keyframe animation created by artists [16] or using biomechanical models of human movement [8, 16].

We add physical simulation as another animation paradigm and allow all paradigms to be used together, both in parallel and sequentially. Our focus is not solely on creating animation using one of these paradigms, but also on

```
<bml>
 <gaze id="gaze1" target="AUDIENCE"/>
 <speech start="gaze1:ready" id="speech1">
  <text>Welcome <sync id="s"/>everybody!</text>
 </speech>
 <gesture id="beat1" hand="LEFT"
          type="BEAT" start="speech1:s1"/>
</bml>
```

Fig. 1 A short example of a BML script, specifying a speech text, and a gesture aligned to the speech

[3] http://hmi.ewi.utwente.nl/showcases/Elckerlyc.

designing a realizer that can make use of—and combine—animation generated by each of them.

2.3 Mixed dynamics

Mixed dynamics combines the precision of kinematic animation (including procedural animation) on certain selected body parts, with the naturalness of physical simulation on the remaining body parts, in a physically coherent manner. It was pioneered by Isaacs and Cohen [5]. Mixed dynamics uses inverse dynamics to determine joint torques on kinematically steered body parts. These torques are transferred to the physically steered body parts. In addition to being affected by joint torques from kinematic joints that are connected to physical body parts, the physical body is affected by gravity, collision impulses, friction forces and joint torques from the physical controllers that steer it. Isaacs and Cohen [5] use a custom designed physics simulator to achieve this. Our mixed dynamics algorithm builds on their ideas, and extends them by using efficient iterative techniques to calculate in real-time the torques exerted by the kinematically steered joints and by providing easy integration with existing real-time physics simulators. However, unlike Isaacs and Cohen's [5] system, we currently require the physical simulation (if active) to act on only one subtree of the skeleton, which must include the root joint. We refer the interested reader to van Welbergen et al. [21] for an extensive overview of real-time animation techniques for mixed dynamics and physical simulation and van Welbergen et al. [22] for a thorough comparison of those methods with ours.

Other hybrid physical simulation/kinematic systems have been designed to switch between full-body kinematic animation and full-body physical simulation and vice versa, depending on the current situations' needs (see [21], Sect. 5.2.4 for an overview). Rather than doing full body switches, we contribute a hybrid method that allows switching to a different mix of physically and kinematically steered joints in real-time.

3 Architecture

We base our architecture on the SAIBA Framework [9], which contains a three-stage process: *communicative intent planning*, multimodal *behavior planning*, resulting in a BML stream, and *behavior realization* of this stream. Elckerlyc encompasses the realization stage.

Running Example 2 (The virtual conductor)
In the virtual conductor system, the intent planning focuses on the musical interaction: *if the musicians play too soft, signal them to play louder*; *if the musicians play very bad, stop the piece*; *if they are too slow, try to correct their timing*; *etcetera*. *The behavior planning uses a many-to-many mapping from intent to behaviors*: *given a communicative intent*, *it selects one of the possible behaviors* (*gesture*, *body movement*, *head movement*) *to express this intent*.

Figure 2 shows our global architecture, and indicates its relation to the overall SAIBA framework. The main information flow goes from the SAIBA Behavior Planning module to the Realizer, in the form of a BML stream. A feedback loop communicates backwards from the Realizer to the SAIBA Behavior Planner. The Realizer notifies the SAIBA Behavior Planner when a behavior is successfully executed and warns it if a particular behavior cannot (or: no longer) be executed. The latter typically occurs when unexpected events occur that prohibit successful realization or, in Elckerlyc, when predictions of user behavior are revised drastically (see Sect. 5.6). The SAIBA Behavior Planner will have to deal with situations where the original plan is no longer feasible, possibly changing its intent, and at least sending an updated BML stream to the Realizer.

We chose to split the realization stage into two parts: *Scheduling* and *Playing*. In the scheduling part, synchronization of the behaviors within and between modalities is determined, and the different behaviors are distributed over appropriate Engines (e.g. Speech Engine, Animation Engine). Each Engine subsequently executes the behaviors for a single output channel (for example: joint rotation for the Animation Engine, sound and local vertex displacements in the face for the Speech Engine). Currently we have implemented an Engine for animation and one for speech with lip synchronization; an Engine for playing other facial animations is in development. The Animation Engine executes all body animation. It automatically combines physical simulation with kinematic animation. We assume that there are no dependencies between the behavior executed by different the Engines other than the time synchronizations maintained by the Peg Board.

3.1 Continuous interaction

Elckerlyc has been designed specifically for *continuous interaction*, in which the behavior of the VH is adapted continually to the behavior of the interaction partner.

Running Example 3 (The virtual conductor)
In the continuous reactive and anticipatory interaction between conductor and ensemble, *the behavior plan that was initially constructed from the score needs to be adapted all the time*. *Some changes merely require re-timing*, *e.g. in order to provide tempo feedback to the ensemble*, *or can be solved simply by adding an extra behavior*: *add an appreciative nod or smile*; *conduct two-handed when the ensemble is*

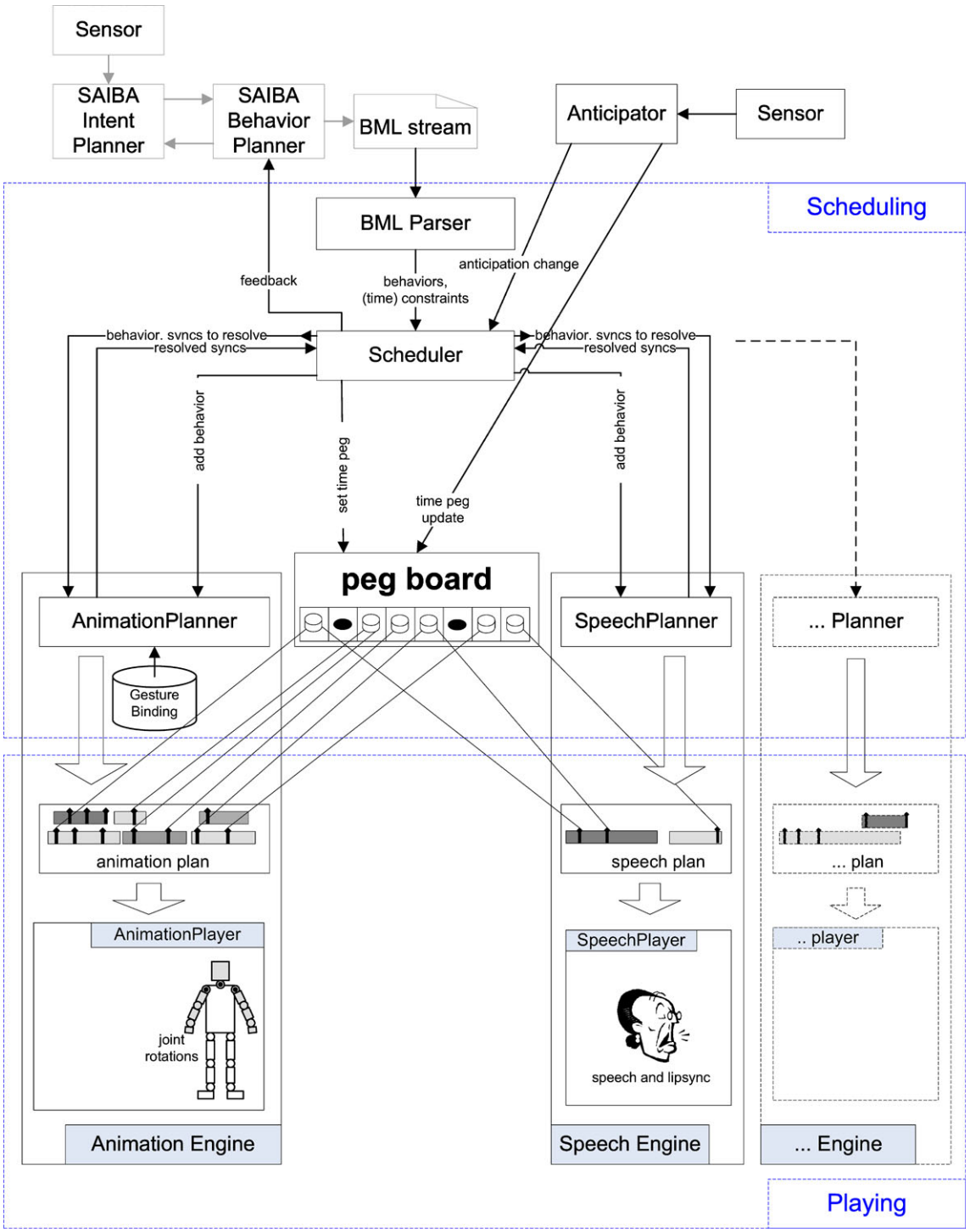

Fig. 2 Architecture of the Elckerlyc BML Realizer, together with the SAIBA Intent Planner and the SAIBA Behavior Planner which make up the other two parts of the SAIBA framework

not paying attention. Other changes are larger and require substantial re-planning. An example of the latter is when the conductor stops in the middle of the piece, because the ensemble completely messed up the music. In this paper we focus on the timing adaptation of ongoing behavior, since that requires specific capabilities in the realizer.

As Running Example 3 suggests, some changes to planned behavior only concern the timing, and should not lead to completely rebuilding the animation plan. Small adaptations of the timing of planned behavior are not specific to conductors, but also occur in other interactions. Elckerlyc offers detailed temporal control over the execution of planned behavior.

To achieve this detailed temporal control, we introduced *Time Pegs* and *Anticipators*. The behavior of our VH is specified in a stream of BML elements (called *behaviors*). Synchronization of the behaviors to each other is done through BML *constraints* that link synchronization points in one behavior (start, end, stroke, etc; see also Fig. 5) to synchronization points in another behavior. We maintain a list of Time Pegs—symbolically linked to those synchronization points that are constrained to be on the same time—on the *Peg Board*, together with the current expectation of their actual execution time (which may change at a later time). Interaction with the world—and conversation partner—is achieved through Anticipators. An Anticipator instantiates synchronization points that can be used in the BML stream to constrain the timing of behaviors. It uses perceptions of events in the real world to update the corresponding Time Pegs, by extrapolating the perceptions into predictions of the timing of future events.

Running Example 4 (The virtual conductor)
For the virtual conductor, an Anticipator has been implemented that predicts the tempo with which the musicians play the music (using audio processing algorithms described elsewhere [14]*). It provides Time Pegs to Elckerlyc that represent the predicted beats. The BML stream for the conductor specifies the appropriate conducting gestures for a piece of music, and their timing. When the musicians play too slow, the conductor would like to increase the tempo. A subtle technique to achieve this is to conduct in the same tempo as the musicians are playing, but slightly ahead of them, so they constantly have the feeling of being 'too late'. This is done by aligning the conducting gestures to the Time Pegs for the predictions of the Anticipator, but slightly ahead of them (see also Fig. 3).*

We have implemented an Anticipator that can predict the tempo of playing music in real-time. For smooth turn-taking purposes, we would like to have an Anticipator that can predict relevant positions to take the turn from user behavior observed through sensors. Designing these kind of Anticipators is challenging, no off-the-shelf systems are available. To demonstrate and test Elckerlyc's continuous interaction capabilities, we have implemented a few simple Anticipators. The Metronome Anticipator, as demonstrated in the webstart, 'predicts' beats of a metronome. The metronome tempo can be adjusted in the user interface while the behavior (linked to its TimePegs) is playing, modifying the timing of this linked behavior in real-time. A Spacebar Anticipator was created for specifying behavior that reacts immidiatly to a spacebar press. Unlike the previously mentioned anticipators, the spacebar anticipator does not provide event prediction, but simply sets the time of a Time Peg to the time of a space bar press or release.

```
<bml id="bml1">
 <gesture id="conduct1" hand="BOTH"
          type="LEXICALIZED" lexeme="3-beat"/>
  <constraint id="c1">
   <synchronize ref="conduct1:start">
    <sync ref="conductingAnticipator:beat1-0.05"/>
   </synchronize>
   <synchronize ref="conduct1:beat2">
    <sync ref="conductingAnticipator:beat2-0.05"/>
   </synchronize>
  </constraint>
</bml>
```

Fig. 3 Example script which uses a conducting anticipator. Two synchronization points in the conduct1 behavior (conduct1:start and conduct1:beat2) are synchronized to desired conducting beats provided by the Conducting Anticipator

3.2 Modularity and extensibility

As in other Realizers [4, 16], modularity and extensibility were explicit requirements in the design of Elckerlyc. We have separated the sometimes complex task of finding the constraints between behaviors described in the BML stream from the actual scheduling of the behaviors. The *BML Parser* is responsible for identifying the constraints between BML behaviors, the *Scheduler* solves these constraints and defines Time Pegs based upon them.[4] The Scheduler then sends the behaviors to the Engine appropriate for that type of behavior. Each Engine consists of a Planner and a Player for a set of BML behavior types. The Scheduler does not need to know about the inner working of an Engine (other than that its Planner implement the Planner interface), it only knows what Engine was registered for certain behaviors. This allows us to easily exchange the Scheduler in order to experiment with different scheduling algorithms. It also allows

[4]The Scheduler does not introduce a novel constraint solver, but for now uses the existing Smartbody scheduling algorithm described in Sect. 5.

us to add or replace Engines easily. We detail our behavior scheduling setup in Sect. 5.

We separate the animation process from the rendering process. The Animation Engine generates motion in the form of joint rotations for each animation frame, that can then be applied to a VH that is rendered in any renderer (in a similar manner as in Thiebaux et al. [16] and Heloir and Kipp [4]). Our mixed dynamics algorithm was designed specifically as a plugin for any existing physics simulator.

Elckerlyc can be used as a black box that converts BML into multi-modal behavior for a VH.[5] If required however, direct access to the Scheduler, Planners, Engines, Plans and Players is also available.

4 The Animation Player

One of the Engines introduced in the previous section is the Animation Engine, consisting of an Animation Planner, that creates an Animation Plan, and an Animation Player (Fig. 4) responsible for executing it. The plan contains descriptions of both physical and procedural motions. Our Animation Player implements and combines several existing state-of-the-art techniques for procedural animation and physical simulation. We introduced the mixed dynamics described earlier [22] in Elckerlyc, to execute the mix of physically and kinematically specified motions simultaneously in a physically coherent manner. In addition, we implemented smooth and automatic switching between physical and kinematic steering (or vice-versa) on parts of the VH's body whenever this particular mix changes. The end result is an integrated animation of the VH which still allows one to adapt the precise timing by adjusting the Time Pegs described in Sect. 3.1.

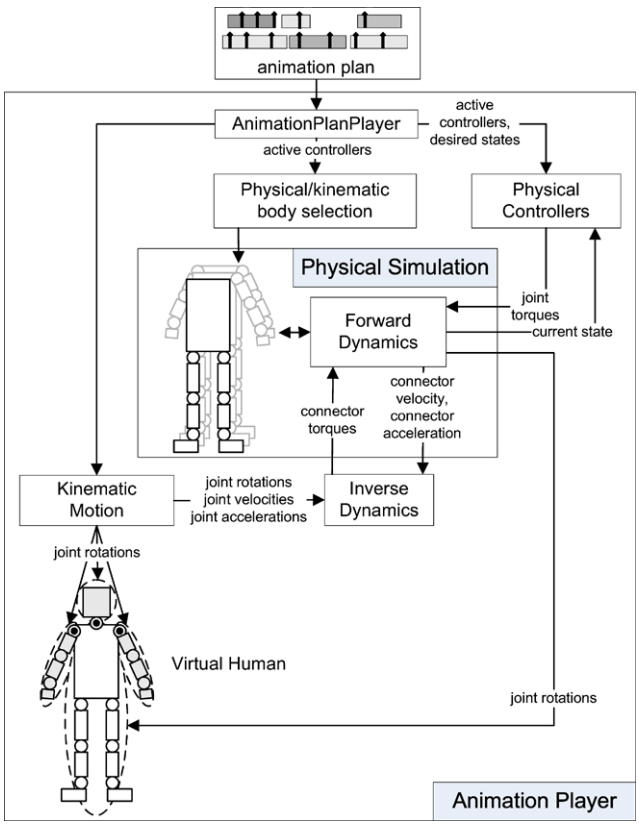

Fig. 4 Animation Player

[5]This functionality can be tested by running the webstart version of Elckerlyc from the web site mentioned earlier.

4.1 The organization of motion

We organize motion in *motion units*.[6] A motion unit has a predefined semantic function (for instance: a three-beat conducting gesture, a walk cycle) and acts on a selected set of joints. A set of parameters can be used to adapt the motion unit (for instance: amplitude for the conducting motion unit, or desired pelvis height and joint stiffness for a physical balancing motion unit). Motion units contain one or more *motion phases*, separated by *keys*. Each key is assigned a predefined canonical time value $0 \leq \alpha_i \leq 1$ ($\alpha_i = 0$ refers to the start of the motion, $\alpha_i = 1$ to its end) that indicates where it is located within the motion unit. Typically a key refers to the (relative) time of a BML synchronization point within the motion unit (see Fig. 5).

Given the current set of parameter values and a canonical time $0 \leq \alpha \leq 1$, a motion unit can be executed, typically by rotating some joints of the VH. We employ a time warping technique to set up the mapping from 'real' time to α (see Sect. 4.4).

4.1.1 *Procedural motion units*

Procedural motion units rotate joints over time as specified by mathematical expressions that take α as well as a vec-

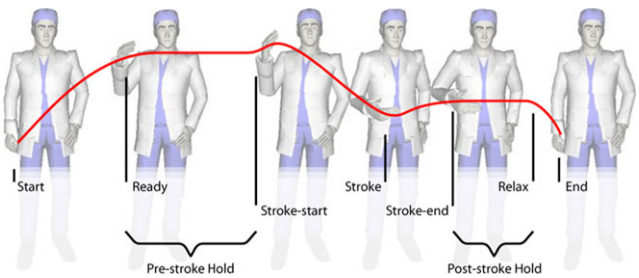

Fig. 5 Standard BML synchronization points (picture from http://wiki.mindmakers.org/projects:bml:main)

[6]Motion structures with a similar function and granularity as our motion unit have been called gestures [3, 13], controllers [16], local motor programs [8], actions [11], or motion spaces [21].

tor $\mathbf{a} \in \Re^n$ as parameters. These expressions can be used to directly steer joint rotation, to position the root or to position the wrists and ankles using analytical inverse kinematics [19]. In Sect. 5.4 we describe how the values of **a** are derived from the BML specification of a behavior.

We support the specification of wrist/ankle positions as continuous mathematical formulas in both global (world) and local (shoulder) coordinate systems. For example, the expression

```
<EndEffector local="false" target="r_wrist"
  translation="0;(1-alpha)*starty+alpha*endy;
  0.3"/>
```

describes how the global position of the right wrist should trace a vertical path, with start and end y position parameters as specified.

Joint rotations can be specified as continuous mathematical functions of α and **a**, or as global or local rotation values defined procedurally (as a function of **a**) at key times (in a similar manner as in [3] and [4]).

The parameter values **a** can be changed in real-time, changing the motion shape or timing. All mathematical expressions are evaluated using the Java Math Expression Parser.[7] Custom function macros can be designed. We defined such macros for Hermite splines, TCB splines [7] and Perlin noise. An example XML-specification of a procedural motion unit can be found at the showcase.[3]

Our design—allowing arbitrary mathematical formulas and parameter sets to be used for motion specification—is more flexible than traditional procedural animation models that define motion in terms of splines or other *predefined* motion formulas and use *fixed* parameter sets [2–4]. Since our design is compatible with these traditional methods, we are able to make use of existing procedural animations. We have semi-automatically converted several motion units from Greta [3] into our XML description for procedural animation. Motion capture animation is also incorporated as a procedural motion unit.

Custom programmed procedural motion units. While our generic procedural motion definition in XML is very flexible, it is sometimes more convenient to author procedural motion units by programming them directly. By doing this, the motion author gains direct access to functionality within the Animation Player. This functionality includes the prediction of a pose at a given time (see Sect. 4.1.3) and provision of the start pose of a motion unit, which makes it very easy to author a motion unit with a flexible start and/or end pose. Biologically inspired gaze and pointing motion units from our previous work [20] were directly converted into such specialized motion units.

[7] Singular Systems, http://sourceforge.net/projects/jep/.

4.1.2 Physical motion units

Physical motion units are executed by physical controllers. Physical controllers use techniques from control theory to steer the VH's 'muscles' in real-time using Newtonian physics, taking friction, gravity, and collisions into account. The input to such a controller is the desired value of the VH's state, for example desired joint rotations or the desired position of the VH's center of mass. The output is a torque applied to one or more joints. To a certain extent, such controllers can cope with and recover from external perturbations.

We have implemented several balancing controllers that offer different trade-offs between balancing stability and movement naturalness [22]. Our pose controllers are simple proportional derivative controllers that loosely keep body parts in their desired position, while still being effected by forces acting on the body.

4.1.3 Transition motion units

Transition motion units are used to create a transition between other motion unit types. They interpolate between the final state (position and velocity) of one motion unit and the predicted initial state of another motion unit. Transition motion units are specified solely by their start and end time and the set of joints they act upon. At animation time, the start pose is taken from the current joint configuration of the VH at the moment that the transition motion unit starts. The end pose is determined by the Animation Predictor. The Animation Predictor uses a copy of the animation plan containing only the *predictable* motion units of the original plan. Predictable motion units are those motion units that deterministically define the pose they set at any given time (for now, only procedural motion units).

We have designed a transition motion unit based upon a slerp transition on each joint and one that creates a C^2 continuous rotation curve between joint rotations [6].

4.2 Mixed dynamics

In many situations, different positive features of procedural motion and physical simulation are needed simultaneously, but acting on different body parts. To achieve this, we dynamically assign a body structure to our VH that is divided in a *physically* steered part and zero or more *kinematically* steered parts. Each part is a tree-structure of joints, connected by rigid bodies. In our current implementation, the physical body part must contain the root joint (or be completely empty). See Fig. 6 for an example set of such body structures.

Running Example 5 (The virtual conductor)
The conducting gesture of the right hand clearly needs the tight temporal control that is best achieved with procedural

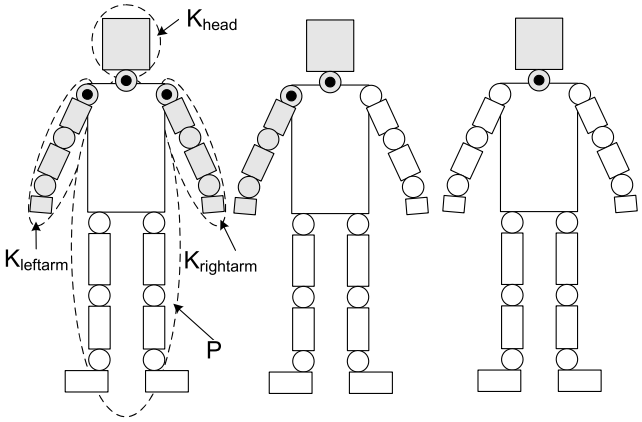

Fig. 6 *Left*: a body divided into kinematic parts K that steer the arms and head and a dynamic physical part P that steers the lower body and trunk. Several variations of K and P are used in the conductor, *this figure* shows three of them

motion. On the other hand, the balanced pose, and the left arm hanging loosely down in gravity, are best left to the realism of physical simulation. The two will necessarily affect each other: the often quite vigorous movements of a conducting motion should have a perceivable impact on the dynamics of the rest of the VH (see also the movies on the showcase).[3]

In our Animation Player (see Fig. 4), motion is executed by motion units which are specified either kinematically or physically. The motion can be adapted in real-time by changing the parameters and timing of a kinematic motion unit or the desired state of a physical motion unit, respectively.

Kinematic motion directly rotates selected joints in the VH. The joint rotations, angular velocities and accelerations of the kinematically steered body parts result in a torque. This torque is calculated using inverse dynamics and applied to the physical body.

The physical controllers in turn apply joint torques that aim at reducing the discrepancy between the desired physical state of the physically steered body part and its current physical state (see Sect. 4.1.2).

Finally, a physics simulator[8] is used to calculate the actual resulting rotations of the physically steered joints. We refer the interested reader to van Welbergen et al. [22] for the implementation details of our mixed dynamics method.

Running Example 6 (The virtual conductor)
For the current gesture repertoire of the virtual conductor, we need exactly the three body configurations shown in Fig. 6: The left-most is used when the VH stands in a balanced pose and makes gestures simultaneously with both hands; the middle one is used for making gestures with the right hand only (letting the left hand hang loosely down in simulated gravity using a physical controller); the rightmost one is needed when the conductor is not gesturing at all.

4.3 Switching physical representation

As can be seen from Running Examples 5 and 6, the specific configuration of kinematically steered and physically steered joints that is active at a certain moment may need to change when the VH performs different behavior. A switch from kinematical to physical control on some body part K is implemented by augmenting the set of physically controlled parts P with the rigid body representation of K and applying the current joint velocity and rotation to the matching joints in the new physical representation. Before the switch, our mixed dynamics approach applied a torque calculated from the movement and position of the joints in K to P. Augmenting P with an articulated set of rigid bodies with joint velocities and rotations that are the same as the joint velocities and rotations in K will obviously result in a similar torque on P. Therefore such a switch results in a smooth transition.

A switch from the physical to kinematic control removes the physical representation of the body part from the physical body of the VH and inserts a new kinematic part K. If the movement on K directly after the switch is similar to the movement in its former physical representation, no sudden forces occur on the new physical body. To achieve this, the kinematic motion unit steering K must smoothly connect to the former physical motion. A simple way to achieve this is by specifying a transition motion unit in BML to connect the physical motion to the kinematic motion. Our Hermite quaternion spline transition motion unit generates a C^2 continuous curve between the end configuration of the joint in the physical body and the start configuration of the kinematic motion that is to be executed on the joint [6]. This curve approximates a rotation path with minimal total angular acceleration from one joint rotation to another, and satisfies the start and end angular velocity constraints prescribed by the physical configuration at the time of the switch and the start configuration of the next kinematic motion unit. Alternatively, one can design a transition motion unit that ensures smooth *end effector* (hand) movement [8] or make use of *procedural* motion units with a flexible start state [3, 8] to ensure a smooth transition.

4.4 Motion execution

The Animation Plan Player takes an animation plan containing instantiations of motion units, called *timed motion units*.

[8] We use the Open Dynamics Engine: http://www.ode.org, but any other real-time physics simulator could be used.

At any specific point in time, it takes the following steps to animate the VH.

(1) *The currently active timed motion units are determined*. The keys of the timed motion units are linked to the Time Pegs, as specified in the BML. Synchronization between keys in different timed motion units is achieved by linking them to the same Time Peg. Each timed motion unit defines a function $f_{mi}(\alpha) = t$ that maps its canonical time α to time t. The active timed motion units are the timed motion units for which $f_{mi}(0) \leq t < f_{mi}(1)$.

Some behaviors (e.g. gaze, posture) allow only one active instance at a time. We support this by assigning the same *replacement group* to all motion units corresponding to such behaviors. Within each replacement group, the active timed motion unit is defined to be the timed motion unit for which $f_{mi}(0) \leq t < f_{mi}(1)$ and $f_{mi}(0)$ has the largest value. If multiple timed motion units within the same replacement group share the same value for $f_{mi}(0)$, the timed motion unit with the smallest $f_{mi}(1)$ is selected as the active timed motion unit for the replacement group.

(2) *Determine which physical body configuration should be assigned*. We infer the continuously changing mix of kinematically and physically steered joints from the active procedural and physical timed motion units. Each physical motion unit specifies on what joints it acts. The selected physical body configuration is the smallest physical body configuration that contains all joints steered by all active timed physical motion units. For example, the behavior planner for the virtual conductor can specify in BML the following behaviors for the left arm: a physical motion unit for ten seconds, that lets the arm hang down, followed by a procedural motion unit to make a certain expressive conducting gesture for one second, and finally again a physical motion unit to let the arm hang down again. To realize this sequence, Elckerlyc automatically executes a switch of the affected joints from physical steering to kinematic steering and back again.

(3) *The active procedural motion units are then executed* at $\alpha = f_{mi}^{-1}(t)$. Most timed motion units calculate $f_{mi}^{-1}(t)$ using a simple linear timewarping scheme [24]. Others use a more complex warping scheme, for instance one to create a biomechanically inspired bell shaped velocity profile in a pointing gesture [20]. Providing mechanisms to combine procedural motion action on the same joint is future work (see also Sect. 6).

(4) *The physical simulation is performed* as a last step. The generated procedural animation is combined with the active timed physical motion units, using the method described in Sect. 4.2 (see also Fig. 4). If different physical controllers act upon the same joint, the sum of their torques is applied to that joint, thus combining their objectives.

5 Scheduling

In the Scheduling stage (see Fig. 2), multimodal behavior described in a BML stream is converted into Plans that are to be executed by Players in different Engines, as already indicated in Sect. 3.2.

5.1 Behavior description

The multimodal behavior that is to be executed by a Planner is described in a BML stream. In addition to supporting standard BML behaviors and the BML synchronization mechanisms, we have defined extra behaviors that specify custom procedural motion units, transition motion units and physical controllers. We had to widen the interpretation of BML's synchronization mechanism in order to allow the specification of synchronization to time predictions of an Anticipator (see Fig. 3). These extensions are documented on our BML Twente (BML$^{\mathrm{T}}$) specification.[9]

5.2 The planners

Each Planner is required to implement functionality to:

1. Add a BML behavior to its Plan.
2. Remove a BML behavior from its Plan.
3. Resolve unknown time constraints on a behavior, given certain known time constraints.
4. Check what (if any) behaviors in the Plan are currently invalid.

Note that a Planner can be queried for time constraints on behavior without adding it to the plan. Potentially, this will allow us to set up more complex schedulers at a later stage that can try out multiple constraint configurations on each behavior. The validity check is typically used to check if a valid plan is retained after an Anticipator moves certain Time Pegs. All implemented Planners check if the order of the Time Pegs of each behavior is still correct (for example: if their start is still earlier than their end). Each Planner can also add validity checks specific to its output modality. These checks are discussed in the section on the specific Planner.

5.3 Scheduling behavior

The Scheduler resolves the timing of behaviors, so that their execution adheres to the *time constraints* specified in the BML. Rather than solving for absolute time stamps for each behavior, the Scheduler assigns each time constraint to a Time Peg and assigns a first prediction of its absolute time. If needed (see Sect. 5.6), these Time Pegs can then be slightly

[9] http://wiki.mindmakers.org/projects:bml:bmlt.

shifted in time, thus moving the absolute timing of the constraint, while keeping inter-behavior synchronization consistent.

The Scheduler has access to several Planners (e.g. Speech Planner, Animation Planner, see also Fig. 2) with which it communicates only through their abstract interface (see Sect. 5.2). It knows for each behavior type which Planner handles it.

Our current scheduling algorithm is based on the SmartBody Scheduler [16]. The behaviors are processed in BML order. The timing of the first behavior is constrained only by its absolute time constraints and by constraints imposed by Anticipators. Subsequent behaviors are timed so that they adhere to time constraints imposed by the already processed behaviors. Our Parser lists all constraints on each behavior. Some of these constraints are resolved by already processed behaviors, but others act on subsequent behaviors in the BML stream and are yet unknown. Our current scheduler delegates resolving these unknown time constraints directly to the Planner of the behavior it is currently processing. Subsequently, the behavior on which all time time constraints are now resolved is added to its Plan.

5.4 Planning animation

The Animation Planner is responsible for selecting motion units based on their BML specification and for inserting them as timed motion units in the animation plan. This plan will then be played by the Animation Player described in Sect. 4. The Planner makes use of the *Gesture Binding* to select motion units. The Gesture Binding is an XML specification describing how a BML behavior is bound to a specific motion unit, possibly constrained by parameters of the BML behavior, and maps parameters in BML to parameters in the motion unit (see Fig. 7). This allows us to bind a new BML behavior to a motion unit without changing Elckerlyc itself and to easily exchange the exact realization of a behavior (by binding it to a different motion unit). Currently, a BML behavior specification is bound to at most one motion unit.

Running Example 7 (The virtual conductor)
Figure 7 shows the BML fragment used to achieve a head nod, and how it is bound to the appropriate procedural head nod animation using the Gesture Binding.

The Animation Planner assists the Scheduler by resolving the execution time of unknown Time Pegs for a motion unit, given certain known time constraints on that motion unit (see Fig. 8). For each procedural motion unit a preferred duration is specified within the definition of the motion unit itself. Time constraints and requested motion times

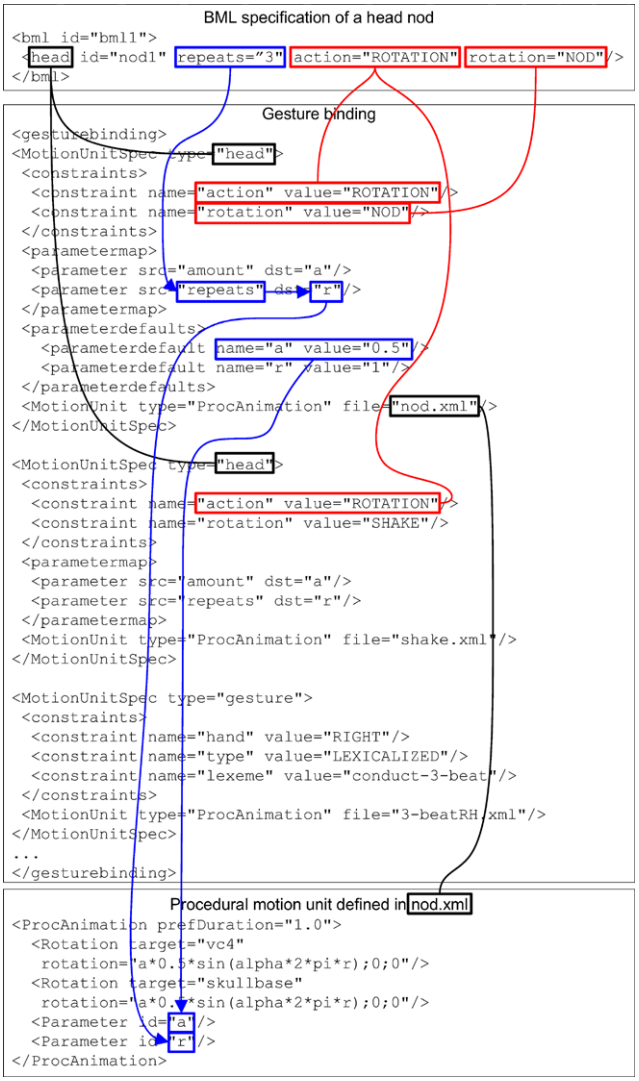

Fig. 7 Gesture Binding fragment binding the <head> element to the nod motion unit. Both the nod and shake motion units execute behaviors of type "head". They both satisfy the constraint action = "ROTATION", but only the nod motion unit satisfies the constraint rotation = "NOD" and is therefore selected to execute the head nod. The Gesture Binding maps the *repeats* parameter value in the BML behavior to the value of parameter *r* specified in the procedural motion unit. The value of parameter *a* is not defined in the BML head behavior, therefore the default value of *a*, as defined in the Gesture Binding, is used in the procedural animation

are linked to the keys of a motion unit. If only *one time constraint* is specified on the motion unit, the requested motion times are resolved in such a way that the specified time constraint is satisfied and the duration of the motion unit is its preferred duration. If *two or more time constraints* are set on a motion unit, uniform scaling is applied to the motion phases between constrained keys. The timing of *start and end keys*, if not constrained, is set to maintain the average stretch or skew (as compared to the preferred duration) caused by the specified time constraints. Note that if *exactly*

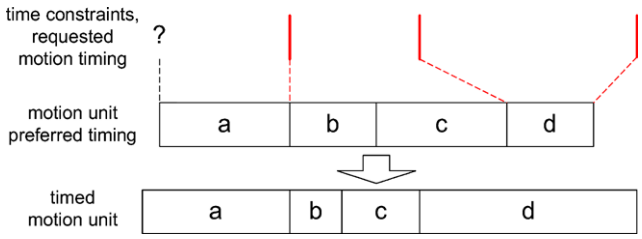

Fig. 8 The Animation Planner is asked to resolve the start time of a behavior with phases *a*, *b*, *c* and *d*, given 3 constraints on its timing. Phase *d* is stretched as prescribed by the two rightmost constraints. Uniform scaling is applied to phases *b* and *c* to satisfy the two leftmost constraints. Phase *a* is slightly stretched to maintain the average stretch resulting from the satisfaction of the time constraints on the behavior

two time constraints act upon the motion unit, the above strategy is equivalent to the uniform scaling used in SmartBody [16]. The current stretching/skewing strategy does not have a biological basis.

Validity checks on behaviors in the Animation Player are Timed Motion Unit specific. For a procedural animation, the minimum and maximum duration are specified, and the validity check fails if the behavior is stretched or skewed beyond these specified limits.

5.5 Planning speech

We execute speech using Microsoft SAPI.[10] Lip synchronization is done by controlling morph targets on the face of the VH. The Speech Planner provides the Scheduler with information on the timing of speech fragments, including the duration of the fragment and the relative start time and duration of words and phonemes. Currently we do not support stretching or skewing of speech. In our current implementation, speech behaviors are marked as invalid if the Time Pegs constraining them enforce a stretch or skew.

5.6 Replanning

The Anticipator notifies the *Scheduler* whenever its predictions change, and updates the Time Pegs. Many of such updates are minor and do not require a change in the Animation Plan or other Plans. Since the keys of timed motion units and timing of speech fragments is symbolically linked to the Time Pegs, the timing update is handled automatically. More significant prediction updates might require re-planning of behavior on several modalities. To check if such an update is needed, the Scheduler asks each Planner if its current plan is still valid. The Scheduler then omits the behaviors that are no longer valid and notifies the SAIBA Behavior Planner using the BML feedback mechanism. It will then be up to the SAIBA Behavior Planner to update the behavior plan (using BML), if desired.

[10] http://www.microsoft.com/speech/.

6 Discussion and future work

We presented Elckerlyc, a modular and extensible BML Realizer designed specifically for continuous interaction. It offers an adjustable trade-off between the control offered by procedural animation and the naturalness enhancements offered by physical simulation. We showed how these capabilities work in practice in our interactive virtual conductor using a running example.

We have assumed that our Engines can operate independently from each other. In practice, some links between Engines might be required, sacrifying some independence for performance or time synchronization reasons. This is a common trade-off in the design of Virtual Human platforms [18]. Currently all our Engines execute their plans timed by the same global clock. To achieve lip sync, the Speech Engine constructs the visemes that are inserted in the face plan of a preliminary version of our Face Engine. We plan to combine these visemes with other facial movement in the Face Engine. The timing of our off-the-shell text to speech system can drift significantly from the the global clock. One implementation of our Speech Engine uses the text to speech system via an intermediate audio file and enforces time constraints by skipping forward and backward in the audio file on significant time drifts (more than 70 ms). This introduces some sound artifacts and still suffers from minor timing differences between speech and lipsync. The Peg Board and Time Pegs are the generic mechanism for synchronization between Engines in Elckerlyc. We plan to achieve finer synchrony and better speech quality by allowing the Speech Engine to run at its own timing and communicate the drift of this timing in relation to the global clock to the other Engines using Time Pegs. Such a design requires only minimal dependencies between Engines (the Speech Engine needs access to the Face Engine to insert visemes in its plan) and makes elegant use of Elckerlycs generic synchronization mechanism: the Time Pegs are used to communicate synchronization constraints between Engines.

An Elckerlyc demo application (as Java webstart) and several demonstration movies and BML scripts can be found on our showcase.[3] We also present our plans for future developments there. Here we present a few of the most important ones.

An important topic for future work is *conflict resolution*. Several behaviors might request the use of the same animation modality, for example the right arm or head. We currently do not resolve such conflicting modality needs within Elckerlyc, although a simple custom conflict solver is used in the SAIBA Behavior Planner of the virtual conductor application. Conflict resolution requires resource allocation (e.g. select the left arm for a gesture if the right arm is doing something else) and resource combination (for instance: combine a gaze with a nod). Resource allocation

could be achieved in Elckerlyc by taking into account the currently used resources when selecting a motion unit from the Gesture Binding. A motion unit can store information on how it might be combined with other motion units [11]. For example, a nod motion should be additively added on top of other head movements [16] and a manipulatory gesture should constrain hand position [4], but might allow elbow movements. The Animation Planner could construct (possibly hierarchical) combinatory motion units that solve resource conflicts [16]. This way, the Animation Plan only contains non-conflicting motion units. Alternatively, the Animation Player could use a final phase that combines conflicting motion units [4]. We give an overview of motion combination techniques for both physical and kinematic motions in van Welbergen et al. [21].

The Scheduler can be improved to select what behaviors to stretch and skew in such a way as to yield the most natural multimodal behavior (given the time constraints in the BML description), rather than selecting behaviors to be stretched and skewed based on where they are located (ordered) in the BML stream, as our (and SmartBody's) current scheduler does. We present some ideas on how to approach this in Nijholt et al. [12]. Within the behaviors we currently use simple linear skewing and stretching. To provide more flexibility we plan to provide formalisms to describe motion unit and motion phase specific stretching/skewing strategies.

Physical simulation in our system is currently limited to one subtree of physical joints of the VH's body, which must contain the root joint. So far, this has not proven to be a restriction in our VH applications. We plan to provide mechanisms to allow a physical joint to be connected to a kinematic parent joint or to a (moving) object in the environment by making use of kinematic constraint mechanisms provided by the physical simulator. This would allow us to use a kinematic root joint and could perhaps help us build a system in which kinematic and physical body can be interleaved more freely. More importantly, this allows interesting interactions with the environment, for example: constrain hand position to a handle to help balancing while riding the bus.

Currently the Gesture Binding provides a one to one mapping from a BML behavior to a motion unit. We plan to allow one to many mapping at a later stage. This could be used to select the best motion unit on the basis of available modalities at its execution time (left hand, right hand) or on the basis of its time constraints (select the motion unit with the shortest preferred duration if fast execution is required) or just to select a random motion unit for e.g. a beat gesture.

Finally, we are currently looking to use Elckerlyc to control a robot head [15] by introducing an alternative Animation Engine.

Acknowledgements The authors would like to thank the Greta and SmartBody developers for allowing us to incorporate their procedural animations in Elckerlyc. We thank Mark ter Maat, who played a role in the early development of Elckerlyc. Elckerlyc's parser/constraint finder was implemented by Ronald C. Paul. This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

References

1. Arnason BT, Thorsteinsson A (2008) The CADIA BML realizer. http://cadia.ru.is/projects/bmlr/
2. Chi DM, Costa M, Zhao L, Badler NI (2000) The EMOTE model for effort and shape. In: SIGGRAPH. ACM Press/Addison-Wesley, New York/Reading, pp 173–182
3. Hartmann B, Mancini M, Pelachaud C (2006) Gesture in human-computer interaction and simulation. In: Implementing expressive gesture synthesis for embodied conversational agents. Lecture notes in computer science, vol 3881. Springer, Berlin, pp 188–199
4. Heloir A, Kipp M (2009) EMBR—a realtime animation engine for interactive embodied agents. In: Intelligent virtual agents. Lecture notes in computer science, vol 5773. Springer, Berlin, pp 393–404
5. Isaacs PM, Cohen MF (1987) Controlling dynamic simulation with kinematic constraints. In: SIGGRAPH. ACM, New York, pp 215–224
6. Kim M, Kim M, Shin SY (1995) A general construction scheme for unit quaternion curves with simple high order derivatives. In: SIGGRAPH. ACM, New York, pp 369–376
7. Kochanek DHU, Bartels RH (1984) Interpolating splines with local tension, continuity, and bias control. In: SIGGRAPH. ACM, New York, pp 33–41
8. Kopp S, Wachsmuth I (2004) Synthesizing multimodal utterances for conversational agents. Comput Animat Virtual Worlds 15(1):39–52
9. Kopp S, Krenn B, Marsella S, Marshall AN, Pelachaud C, Pirker H, Thórisson KR, Vilhjálmsson HH (2006) Towards a common framework for multimodal generation: the behavior markup language. In: Intelligent virtual agents. Lecture notes in computer science, vol 4133. Springer, Berlin, pp 205–217
10. Loyall AB, Reilly WSN, Bates J, Weyhrauch P (2004) System for authoring highly interactive, personality-rich interactive characters. In: Symposium on computer animation. Eurographics Association, Munich, pp 59–68. doi: 10.1145/1028523.1028532
11. Neff M, Fiume E (2008) From performance theory to character animation tools. In: Human motion—understanding, modelling, capture, and animation. Springer, Dordrecht, pp 597–629
12. Nijholt A, Reidsma D, van Welbergen H, op den Akker HJA, Ruttkay ZsM (2008) Mutually coordinated anticipatory multimodal interaction. In: Nonverbal features of human-human and human-machine interaction. Lecture notes in computer science, vol 5042. Springer, Berlin, pp 70–89
13. Noot H, Ruttkay Zs (2005) Variations in gesturing and speech by gestyle. Int J Hum Comput Stud 62(2):211–229
14. Reidsma D, Nijholt A, Bos P (2008) Temporal interaction between an artificial orchestra conductor and human musicians. Comput Entertain 6(4):1–22
15. Reilink R, Visser LC, Bennik J, Carloni R, Brouwer DM, Stramigioli S (2009) The Twente humanoid head. In: IEEE international conference on robotics and automation, pp 1593–1594
16. Thiebaux M, Marshall AN, Marsella S, Kallmann M (2008) Smartbody: behavior realization for embodied conversational agents. In: Autonomous agents and multiagent systems, pp 151–158

17. Thórisson KR (2002) Natural turn-taking needs no manual: computational theory and model, from perception to action. In: Multimodality in language and speech systems. Kluwer Academic, Dordrecht, pp 173–207
18. Thórisson KR, Benko H, Abramov D, Arnold A, Maskey S, Vaseekaran A (2004) Constructionist design methodology for interactive intelligences. AI Mag 25(4). doi: 10.1609/aimag.v25i4.1786
19. Tolani D, Goswami A, Badler NI (2000) Real-time inverse kinematics techniques for anthropomorphic limbs. Graph Models Image Process 62(5):353–388
20. van Welbergen H, Nijholt A, Reidsma D, Zwiers J (2006) Presenting in virtual worlds: towards an architecture for a 3D presenter explaining 2D-presented information. IEEE Intell Syst 21(5):47–53
21. van Welbergen H, van Basten BJH, Egges A, Ruttkay ZsM, Overmars MH (2009) Real time character animation: a trade-off between naturalness and control. In: Pauly M, Greiner G (eds) Eurographics—state of the art reports. Eurographics Association, Munich, pp 45–72. ISSN 1017-4656
22. van Welbergen H, Zwiers J, Ruttkay ZsM (2009) Real-time animation using a mix of physical simulation and kinematics. J Graph GPU Game Tools 14(4):1–21
23. Vilhjálmsson HH, Cantelmo N, Cassell J, Chafai NE, Kipp M, Kopp S, Mancini M, Marsella SC, Marshall AN, Pelachaud C, Ruttkay ZM, Thórisson KR, van Welbergen H, van der Werf RJ (2007) The behavior markup language: recent developments and challenges. In: Intelligent virtual agents. Lecture notes in computer science, vol 4722. Springer, Berlin, pp 99–120
24. Witkin A, Popovic Z (1995) Motion warping. In: SIGGRAPH. ACM, New York, pp 105–108