An Incremental Multimodal Realizer for Behavior Co-Articulation and Coordination

Herwin van Welbergen[1], Dennis Reidsma[2], and Stefan Kopp[1]
{hvanwelbergen,skopp}@techfak.uni-bielefeld.de
d.reidsma@utwente.nl *

[1] Sociable Agents Group, CITEC, Fac. of Technology, Bielefeld University
[2] Human Media Interaction, University of Twente

Abstract. Human conversations are highly dynamic, responsive interactions. To enter into flexible interactions with humans, a conversational agent must be capable of fluent incremental behavior generation. New utterance content must be integrated seamlessly with ongoing behavior, requiring dynamic application of co-articulation. The timing and shape of the agent's behavior must be adapted on-the-fly to the interlocutor, resulting in natural interpersonal coordination. We present AsapRealizer, a BML 1.0 behavior realizer that achieves these capabilities by building upon, and extending, two state of the art existing realizers, as the result of a collaboration between two research groups.

1 Introduction

Human conversations are highly dynamic, responsive interactions. In such settings, extended utterances are not pre-planned and then executed, but are produced by a speaker incrementally. This incremental delivery enables the speaker, first, to decompose a complicated intended message into a series of fragments and, second, to perceive the addressee's behavior and to adapt a running contribution in order to ensure the success of the communicative activity. Notably, such utterances still exhibit a high degree of synchrony between multimodal behaviors like speech, gestures, or facial expressions [1]. One mechanism with which incremental production of utterances is facilitated is co-articulation between adjacent behaviors. Co-articulation, for example, occurs when the retraction phase of one gesture and the preparation phase of the next one become fused into a direct transition [2]. In this way, co-articulation helps to create (or is a consequence of) fluently connected utterances and increases the flexibility for creating multimodal synchrony.

At the same time, conversation between humans is characterized by interpersonal coordinations on several levels. Bernieri et al. [3] define interpersonal coordination as the degree to which behaviors in an interaction are non random, patterned or synchronized in both timing and shape. They categorize interpersonal coordination in behavior matching or similarity and interactional

* This work was partially supported by the DFG in the Center of Excellence "Cognitive Interaction Technology".

synchrony. Behavior matching includes mimicry such as interlocutors adapting similar poses. Interactional synchrony includes alignment of movement rhythm (e.g. alignment of postural sway or breathing patterns), synchronization of behavior (e.g. simultaneous posture changes by a listener and speaker) and smooth meshing/intertwining of behavior (in conversation these are, e.g., smooth turn-taking and backchannel feedback such as head nods or uttering "uh huh").

We aim to develop conversational agents that can make use of all these features of human conversational behavior: multimodal synchrony, flexible interpersonal coordination, and fluently connected, incrementally produced utterance with natural co-articulation. Thus far, no single virtual human generation platform features all of these capabilities. Therefore, we have developed AsapRealizer, a BML behavior realizer for virtual humans that builds on two existing realizers that have focused on either incremental multimodal utterance construction [4] or interactional coordination [5] as isolated problems. By combining the realization capabilities for incremental multimodal behavior construction and interactional coordination in a single realizer, we can enable interaction scenarios that go beyond the capabilities of these individual realizers.

A key feature of AsapRealizer is its ability to continuously and automatically adapt ongoing behavior while retaining its original specification constraints. This affords incremental processing in two ways: First, the behavior planner can issue behavior requests early on and then send detailed parameters later on. Second, the realizer can rely on predicted events and then adapt as new, possibly altered status information about these events arrive. An example of the former is a behavior that needs to be fluently connected to an ongoing behavior and might partly co-articulate with it, or when a sudden parameter change (e.g. increase in gesture amplitude) results in changed shape and timing of the ongoing and subsequent gesture phases. An example of the latter is when a behavior is synchronized to an external event (e.g. the interlocutor's head nod), or when behavior execution cannot be reliably predicted (as with a robotic body [6]). In case of updates to the timing of this behavior, continuous adjustment of other, synchronized, modalities may be needed to achieve specified time constraints. The AsapRealizer provides a way to cope with these challenges by interleaving scheduling and execution freely. This paper presents AsapRealizer's design and implementation, explains how it allows for incremental adaptive scheduling of behavior, and demonstrates how its implementation is applied specifically to achieve gestural co-articulation.

2 Related Work

It is increasingly acknowledged that, to achieve a more natural dialog, social agents require incremental (dialog) processing [7]. Such incremental processing enables social agents to exhibit interpersonal coordination strategies such as backchannel feedback, smooth turn-taking, or an alignment of movement rhythm between interlocutors. One prerequisite for this is a flexible behavior

generation algorithm that is able to change the timing and shape of ongoing behavior on the fly and in *anticipation* to the movement of an interlocutor [8].

Most existing behavior realizers comply with the SAIBA Framework [9], in which the BML markup language 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 the relative timing of the involved actions. Synchronization among behaviors is done through BML constraints, included within a BML block, that link synchronization points in one behavior (like “start”, “end”, “stroke”, etc; see also Fig. 1) to similar synchronization points in other behaviors.

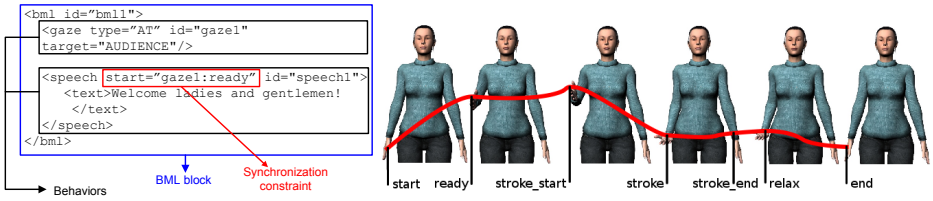

Fig. 1: Left: an example of a BML block. Right: the standard synchronization points of a gesture.

Several BML realizers have been implemented by different research groups [10,11,12,13], while other research groups have expressed their intent to join the SAIBA effort (e.g. the authors of [4,14]). Of these realizers, ACE [4] and Elckerlyc [13] are especially relevant for AsapRealizer, because unlike the other realizers, they are specifically designed to allow incremental behavior construction and interactional coordination, respectively.

For the present work, it is necessary to fluently stream the execution of behaviors from different BML blocks. However, a specification for co-articulation constraints between subsequent BML blocks is currently lacking: BML blocks can be only specified to start instantly, merging with ongoing behavior, or to start after all currently ongoing behavior blocks are completely finished. Furthermore, while BML provides a clear-cut specification of the internal multimodal synchronization of the behavior of a virtual human, it lacks the expressivity to specify the interaction of this behavior with other (virtual) humans. AsapRealizer provides a BML extension *BMLa* to address these shortcomings. To this end, BMLa adopts Elckerlyc’s interactional coordination specification mechanisms (see below). In addition to that, BMLa provides novel mechanisms to specify the combination of BML blocks. These mechanisms provide a Behavior Planner or other authors of a BML stream with specific control over whether or not co-articulation between gestures or other behaviors in two or more BML blocks may occur.

The ACE (Articulated Communicator Engine) [4] realizer was the first behavior generation system that simulated the *mutual* adaptations between the

timing of gesture and speech that humans employ to achieve synchrony between co-expressive elements in those two modalities. It also pioneered the incremental scheduling of multimodal behavior for virtual humans. ACE's incremental speech-gesture production model is based on McNeill's segmentation hypothesis [2]: speech and gesture are produced in successive *chunks*. Each chunk contains one prosodic phrase in speech and one co-expressive gesture phrase. Gesture movement between the strokes of two successive gestures (in two successive chunks) depends on their relative timing, ranging from retracting to an in-between rest position, to a direct transition movement. A flexible silent pause in speech was inserted to create enough time for the preparation phase of the second gesture. To achieve this production flexibility, ACE uses an incremental scheduling algorithm that plans part of the chunk in advance and refines it when the chunk is actually started, by setting up each chunk's inter-chunk synchrony with its predecessor. This starting time is decided in a bottom-up process, where the current hand and body positions influence the actual duration and trajectory of the preparation and retraction phases of two adjacent gestures. There is no continuous adaptation in ACE after the initiation of a chunk.

Elckerlyc [5], a more recent realizer, pioneered new ways of BML specification (using its BML extension BML^T) and an implementation of several behavior generation mechanisms that are essential for interactional coordination. These mechanisms include graceful interruption, re-parameterization of ongoing behavior and synchronization to predicted interlocutor behavior. For this, Elckerlyc provides a flexible behavior plan representation that can continuously be modified, while retaining the constraints specified in BML. The construction and representation of Elckerlyc's multimodal plan is discussed in detail in [5]. Thus far, updates to Elckerlyc's multimodal plan were mostly guided through top-down processes (e.g. by specifying them in BML) and by directly aligning the timing of synchronization points to (maybe predicted) interlocutor events.

In sum, the two different realizers provide useful concepts for enabling the kind of flexibility needed to simulate natural coordination and co-articulation in behavior realization. AsapRealizer builds upon both approaches, combines them into a coherent architectural framework and adds new concepts. AsapRealizer's design generalizes ACE's incremental scheduling mechanism to one that supports behavior specification in BML blocks. It further provides the ability to do bottom-up adaptation of ongoing behavior. For example, AsapRealizer implements the bottom-up gesture modification that is employed in ACE, and combines it with Elckerlyc's flexible plan representation and its approach to enable the specification and implementation of interactional coordination.

3 Design Considerations

AsapRealizer should enable both inter-personal coordination and inter-behavioral synchrony through incremental behavior construction and flexible scheduling. As a realizer component within the SAIBA framework, it should be easy to use

in many virtual human applications and experiments. To achieve this, AsapRealizer's design satisfies the following requirements:

1. Generate multimodal behavior specified in BML.
2. Generate behaviors incrementally and link increments fluently, with natural co-articulation between the increments.
3. Process underspecified BML behavior specifications that constrain only those features the author or Behavior Planner is really interested in achieving; that is, keeps all valid realization possibilities open for as long as possible. Figure out unspecified timing or shape (e.g. trajectory, hand shape, amplitude) of a motor behavior in a biologically plausible way.
4. Allow last minute changes in shape and timing of behavior, even when the behavior is currently ongoing; check for and maintain validity of the constraints specified in BML.
5. Enable top-down (through BML) and bottom-up processes (e.g. changing predictions, co-articulation with new behavior) to adapt a behavior.

We make use of several design elements and implementations from both Elckerlyc and ACE to satisfy these requirements. Req. 1 is satisfied by building AsapRealizer on top of the Elckerlyc BML Realizer. To satisfy Req. 2, we have designed novel algorithms for both the specification (in BMLa) and the implementation of incremental construction of behavior using BML blocks. In Section 5.1 we explain these algorithms in detail. Req. 3 is behavior-specific. We adopt mechanisms from ACE to automatically construct preparation and retraction phases of gestures that provide biologically plausible timing. The timing of these gesture retractions and preparations is updated on the fly. Req. 4 and 5 are satisfied by combining Elckerlyc's flexible behavior plan representation with ACE's behavior and chunk state management. The latter allows flexible bottom-up changes of the ongoing behavior plan, while the first assures that all adaptations are subject to the constraints specified in BML. The implementation of this functionality is discussed in Section 4. AsapRealizer also supports all of Elckerlyc's top-down behavior adaptations to achieve interpersonal coordination.

Fig. 2 illustrates how we have incorporated design features from ACE and Elckerlyc into AsapRealizer. We make use of BML to specify behavior, enhanced to allow the specification of whether or not behavior co-articulation may occur between BML blocks. The scheduling of specified behaviors results, as in Elckerlyc, in a flexible behavior plan representation –the PegBoard– that allows one to do timing modifications to the behavior plan in such a way that the BML constraints remain satisfied and no expensive re-scheduling is needed. An ExecutionEngine executes the constructed plan and, like ACE, continuously makes modifications to this ongoing plan. These modifications are split into shape modifications that modify the form of behaviors and time modifications that directly act upon the PegBoard. These bottom-up time modifications thus do not invalidate the time constraints as specified in BML. In addition, like Elckerlyc, AsapRealizer can align (and continuously update) the timing of behavior to anticipated events and exert top-down plan or behavior modifications (e.g. interrup-

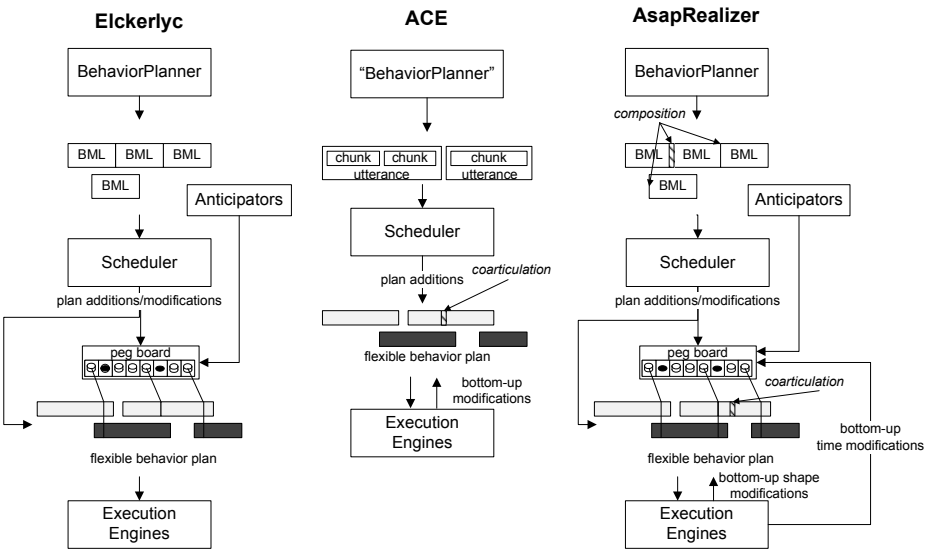

Fig. 2: The design of Elckerlyc, ACE and AsapRealizer

tion and parameter changes in ongoing behavior). Finally, we have implemented a novel gesture co-articulation strategy that extends the strategy used in ACE.

4 State-Based Behavior Scheduling

AsapRealizer realizes a stream of BML blocks, each of which specifies the timing (e.g. sync points X of behavior A and Y of behavior B should occur at the same time) and shape (e.g. behavior A should be performed with the left hand) of the desired behaviors. Generally, BML blocks are under specified and leave realizers freedom in their actual realization. Realizers can make use of this to achieve natural looking motor behavior, e.g. by setting a biologically plausible duration of a gesture preparation. However, most realizers [10,14,11,12] exploit this freedom only for behavior plan construction/scheduling. After scheduling, no more changes can be made and the plan is executed ballistically. AsapRealizer employs Elckerlyc's plan representation [5] that allows changes even when being executed, while retaining the specified BML constraints.

The generation of a BML block and its individual behaviors are managed by two state machines as shown in Fig. 3. These state machines are adopted from Elckerlyc but extended according to the phases of ACE's incremental production model: The behavior state machine (Fig. 3, left) 1) adds a SUBSIDING state, and 2) supports continuous bottom-up adaptation of ongoing behavior using the `updateTiming` function. The BML block state machine (right) 1) provides a SUBSIDING state and 2) has a mechanism to delay the start of a BML block until all its chunk targets are either retracted or finished. These two machines work

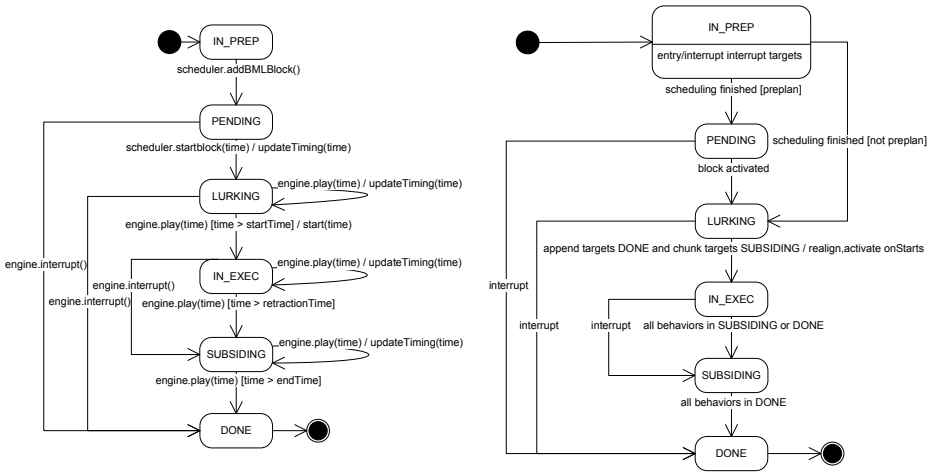

Fig. 3: The behavior (left) and BML block (right) state machines in AsapRealizer.

as follows: Behaviors start out in the IN_PREP state. Once all behaviors of a BML block are scheduled, they move into the PENDING state. This transition is triggered by the central scheduler. At the same time, the block machine moves into the LURKING state. BML blocks can contain ordering constraints that require them to start (fluently) after other blocks. Once these constraints are satisfied, the block moves into the IN_EXEC state. The scheduled plan might already not be the most suitable anymore since the behavior context (e.g. hand position, resting posture) might have changed during scheduling or while waiting for other blocks to finish. Therefore this state transition triggers a light-weight behavior realignment step, in which the timing of each behavior is re-evaluated.

Once the new timing is set, all behaviors in the block move to the LURKING state. Behavior state updates are then managed in a bottom-up fashion, through the playback loop of a specific execution engine (e.g. SpeechEngine, AnimationEngine, FaceEngine). Within an animation loop, each playback step is generally executed on an Engine by calling its play function with the current time. This first invokes a timing and shape update on the behavior, using the `updateTiming` function. If the start time of the behavior is greater than the current time, the behavior will be started, moving it to IN_EXEC state. Some behaviors (e.g. gestures) contain a SUBSIDING state. The SUBSIDING state is held while moving the behavior back to a resting position after some meaningful motor behavior was executed within its IN_EXEC state. During the IN_EXEC and SUBSIDING stages, the behavior is executed on the virtual character. While the behavior is being executed, its shape and the timing of its sync points are continuously updated using its `updateTiming` function. Once its end time is reached, the behavior moves to the DONE state. The IN_EXEC, SUBSIDING and DONE phases of a BML block represent the cumulative state of all behaviors in the block. A BML block enters the SUBSIDING state when all of its behaviors are either SUBSID-

ING or DONE and moves to the DONE state when all its behaviors are DONE. Behaviors and BML blocks may (gracefully) be interrupted at any time after they are scheduled.[3] Interruption can be triggered both top-down (e.g. by specifying in new BML blocks that certain behaviors must be interrupted) or bottom-up (e.g. when the execution of a behavior fails). When behaviors are interrupted they move into their SUBSIDING phase and are gracefully retracted. The exact implementation of this retraction is behavior-specific; Section 5.2 discusses the implementation used for gesture.

5 Results

AsapRealizer adheres to the new BML 1.0 standard.[4] Compliance to this standard is tested using the RealizerTester framework [15]. We have supported this compliance testing effort by providing several new BML 1.0 test cases.

Through the combination of key features from ACE and Elckerlyc, AsapRealizer provides two main capabilities that go beyond other realizers. Firstly, AsapRealizer improves upon ACE by providing more generic co-articulation mechanisms. Section 5.1 explains how this co-articulation can be specified using BML, as well as how the mechanisms are implemented. Secondly, AsapRealizer provides novel, highly flexible capabilities for specifying and executing graceful interruption of ongoing behaviors, as discussed in Section 5.2.

5.1 Simulating Gesture Co-articulation

AsapRealizer's state-based scheduling and planning allow for simulating interactions between successive behaviors. This, in addition to the interpersonal coordination capabilities of Elckerlyc, makes it suitable for the fluent incremental generation of behavior in which natural co-articulation effects emerge. In this section we demonstrate how the implementation and specification of gesture co-articulation (as shown in Fig. 4) is achieved within our architecture.

Specifying Gesture Co-articulation in BML The occurrence of gesture co-articulation (or the lack thereof) can well have a communicative function (e.g. marking information boundaries) [1] and is not a matter of simply 'gluing together gestures' in a realizer. Therefore, we allow the Behavior Planner to have control over whether or not gesture co-articulation should occur. This means that some manner of expressing gesture co-articulation has to be provided in BML. We achieve this using the BML `composition` attribute, which allows for merging a BML block into the ongoing behavior plan (i.e. starting it instantly) and for appending a BML block to the behavior plan (i.e. starting it after *all*

[3] AsapRealizer also provides functionality to forcefully stop a behavior or BML block at any time. This functionality is mainly used to exit or reset the realizer. For clarity reasons it is not show in Fig. 3.

[4] `http://www.mindmakers.org/projects/bml-1-0/wiki`

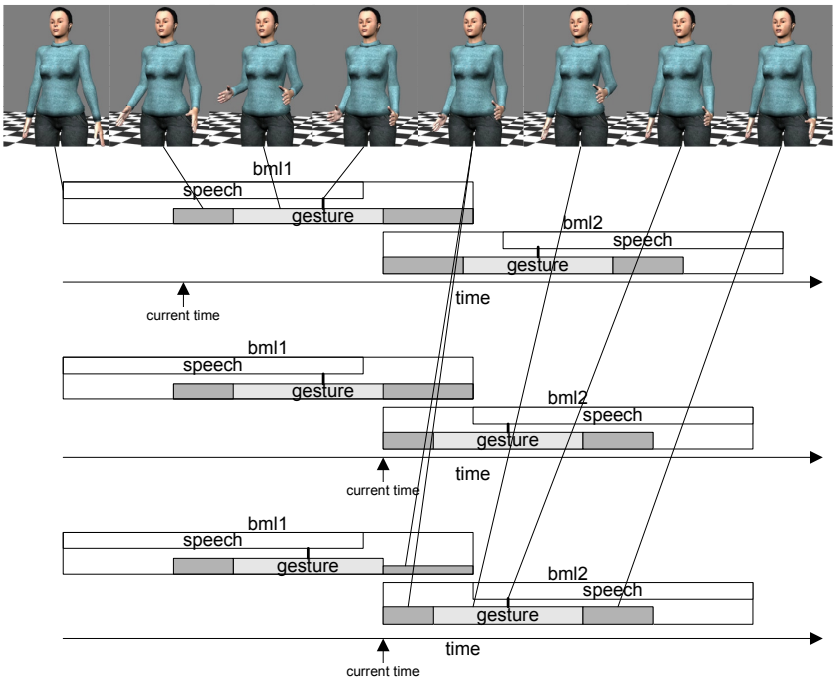

Fig. 4: An example of gesture co-articulation: First BML block `bml1` is being executed and a preliminary plan for `bml2` is being created (top plan graph). As `bml1` is subsiding, `bml2` is re-aligned to fit the current behavior state (middle). This involves shortening the gesture preparation since the hand is still in gesture space. As the gesture of `bml1` is being retracted, it has a lower priority than the preparation of the gesture of `bml2` and is overridden by it (bottom plan graph). Since `bml2`'s gesture acts only on the left hand, a cleanup motion is generated for the right hand part of `bml1`'s gesture.

ongoing behavior). BMLa adds two new composition attribute values that allow us to specify the relation of our block with the ongoing behavior plan in more detail: `append-after(X)` and `chunk-after(X)`. `append-after(X)` specifies the BML block to start after all behavior in the set of BML blocks X is finished. `chunk-after(X)` specifies the BML block to start as soon as all behavior in the set of BML blocks X are either finished or in retraction (i.e. the blocks are SUBSIDING). BML Example 1 illustrates the use of this attribute.

BML Example 1 Expressing gesture co-articulation in BML.

```
<bml id="bml2" composition="chunk-after(bml1)">
  <speech id="speech1">
    <text>At <sync id="s1"/>6 pm you have another appointment</text>
  </speech>
  <gesture id="gesture1" lexeme="BEAT" stroke="speech1:s1"/>
</bml>
```

Implementation of Gesture Co-articulation To allow gesture co-articulation between BML blocks, a realizer needs information on when the new BML block can be started and requires an animation system that allows a new gesture to fluently overtake a retracting old gesture. To fulfill the first requirement, AsapRealizer keeps track of each BML block's state using the block state machine (Fig. 3). The state transition from LURKING to IN_EXEC is triggered for a new BML block only if all chunk targets of the block are either SUBSIDING or DONE.

AsapRealizer's AnimationEngine builds upon Elckerlyc's mixed dynamics capabilities [16], allowing a mix of the physical realism provided by physical simulation and the control (in timing and limb placement) provided by procedural animation or motion capture. These capabilities are integrated with the bottom-up re-planning and adaptation provided by ACE. In addition, AsapRealizer provides a novel animation conflict resolution solver that employs a dynamic resting state rather than specifying the resting state only implicitly using the now deprecated BML behavior persistence (as in Elckerlyc) or as a preset joint configuration (as in ACE). Here we highlight how this functionality is employed for gesture co-articulation.

In AsapRealizer's AnimationEngine each behavior is executed using a TimedMotionUnit (TMU, henceforth), which specifies (among other things) the state of its behavior, a priority and the set of skeletal joints it controls. Whenever a TMU is played back, it can set joint rotations, apply a physical controller to the physical part of the body model, or set a new RestingTMU. The latter is a special TMU that creates motions leading into and managing a dynamic 'resting state' of the virtual human. Implementations of a RestingTMU could, e.g., model balanced lower body movement using a physical balance controller or slightly move the body using Perlin noise. There is only one RestingTMU and it is always

executed with the lowest priority, i.e., the other TMUs take precedence over it. Such a precedence may be partial (e.g. only on limbs steered by other TMUs).

Gesture co-articulation is achieved using a conflict resolution mechanism within the AnimationEngine. TMUs that execute gesture behaviors automatically reduce their priority when the behavior enters the retraction phase. The AnimationEngine executes TMUs in the order of their priority. Whenever a TMU needs to control joints that are already controlled by higher priority TMUs, this specific TMU is interrupted by the AnimationEngine. Such a TMU might however have previously exerted control upon joints that are not taken over by the higher priority TMUs. These joints need to be gracefully moved back to their resting state. This is automatically taken care of by the AnimationEngine: a new, low priority 'cleanup' TMU is created and inserted in the animation plan.

Bottom-up adaptive timing is crucial in gesture preparation and retraction, since the start position of the preparation, the hand position at the start/end of the stroke and the posture state at the end of the gesture are all subject to change during gesture execution. The hand position may vary by previously executed motion and/or posture changes, the hand position at the start/end of the stroke may vary by parameter adjustments in the gesture, and the rest posture state may change during execution. We have implemented an adaptive timing process for the preparation and retraction of gestures. It makes use of Fitts' law to dynamically determine a biologically plausible duration of the hand movement trajectory from the current position to the hand position at the start of the stroke phase (for the preparation), or from the end of the stroke to the current rest position (for the retraction).

5.2 Graceful Interruption

AsapRealizer allows one to specify graceful interruption of ongoing behavior, including, when desired, replacement behavior that is fluently connected to the interrupted behavior. The mechanisms used for this are based on the handling of graceful interruption in Elckerlyc [17]. In Elckerlyc, this was handled completely in a top-down fashion (see also BML Example 2). Assuming we want to interrupt a BML block `bml1` containing some speech and a gesture `gesture1`, a Behavior Planner using Elckerlyc had to adapt the following interruption strategy:

1. If `gesture1` did not start yet, interrupt everything in `bml1` (BML Example 2a).
2. If `gesture1` is already finished or retracting, there is no need to interrupt it; only interrupt all other behaviors in `bml1` (BML Example 2b)
3. If `gesture1` is currently being executed, replace it by a movement that moves it back to the rest pose (BML Example 2c)

This verbose interruption strategy requires the Behavior Planner to have knowledge on the state the gesture is in, on the current resting state of the virtual human and on how a graceful interruption behavior can be selected that moves the virtual human towards this state. In AsapRealizer, all this knowledge

BML Example 2 The specification of graceful interruption in Elckerlyc.

(a) Interrupt all behavior in `bml1`

```
<bml id="yieldturn">
  <bmlt:interrupt id="i1" target="bml1"/>
</bml>
```

(b) Interrupt all behavior in `bml1` excluding `gesture1`.

```
<bml id="yieldturn">
<bmlt:interrupt id="i1" target="bml1" exclude="gesture1"/>
</bml>
```

(c) Interrupt all behavior in bml1. Insert a behavior (relaxArm) that gracefully moves the gesturing arm back to its rest position.

```
<bml id="yieldturn">
  <bmlt:interrupt id="i1" target="bml1"/>
  <bmlt:controller id="relaxArm" class="CompoundController"
    name="leftarmhang"/>
</bml>
```

and functionality is available in the AnimationEngine. Therefore, the Behavior Planner can achieve graceful interruption using BML Example 2a, regardless of the state the gesture is in. This interrupt request is then handled by the AnimationEngine, which generates an automatic retraction motion to the rest pose (using functionality provided by the RestingTimedMotionUnit), if needed. When full control over the exact retraction motion is required, retraction motions can still be specified manually. Such a specified retraction will have a higher priority than the automatically generated one, and as such simply overrule it. The interrupted gesture (that is now in its retraction phase) can also be overwritten by a new gesture, using the AsapRealizer's gesture co-articulation mechanisms.

In summary, the combination of ACE's co-articulation and bottom-up adaptation capabilities, Elckerlyc's top-down interruption specification and any-time adaptability, and the new dynamic pose specification, gives AsapRealizer a novel, highly flexible mechanism for specifying and executing graceful interruption of ongoing behavior, and (when desired) insertion of new co-articulated gestures.

6 Discussion

We have introduced AsapRealizer, a new, BML 1.0 compliant –as tested using the RealizerTester framework [15]– realizer. AsapRealizer's unique capability to continuously and automatically adapt ongoing behavior while retaining its original specification constraints makes it eminently suitable for virtual human applications that require interactional coordination and incremental, fluent behavior generation. Its flexibility is achieved by implementing a fusion of the state of the art multimodal behavior generation features of ACE and Elckerlyc. In this paper, we illustrated how AsapRealizer goes beyond other realizers by discussing

its more generic co-articulation mechanisms and its novel, highly flexible capabilities for specifying and executing graceful interruption of ongoing behaviors. AsapRealizer allows us to realize interaction scenarios that go beyond the capabilities of each the individual realizers.

The co-articulation mechanism generalizes ACE's incremental generation of chunks into a mechanism that uses BML blocks as increments instead of chunks –which can also describe many other synchronization possibilities. The interruption scenario illustrates a capability for highly responsive interaction that cannot be realized through either of the contributing systems alone.

Another scenario of responsive interaction enabled by the *combination* of ACE and Elckerlyc is that of interactional coordination. Elckerlyc, and therefore AsapRealizer as well, provides BML specification mechanisms to synchronize the behavior of the virtual human to (anticipated) time events in interlocutor behavior. This functionality has been used to make micro-adjustment to the timing of behavior, for example to align the movement of a virtual fitness trainer to that of the user she exercises with [18], or to delay the start of an utterance after receiving user feedback in an attentive speaker [17]. There are other interaction coordination scenarios that cannot be satisfied by Elckerlyc's time adjustment mechanism alone, but also require (bottom-up) shape adjustments and/or the insertion of new behavior segments. For example, the virtual human could point at an object, wait for the interlocutor to gaze at this object –achieving joint attention– and then retract the pointing gesture and continue speaking. This requires the insertion of a hold motion if the user is not yet gazing at the object as the gesture finishes its stroke, and the automatic, fluent, continuation after a hold motion once joint attention is achieved.[5] Such larger plan adaptations cannot easily be realized with Elckerlyc, since Elckerlyc requires the content and timing of any adaptation to be fully specified in a top-down fashion by the Behavior Planner. AsapRealizer can use its bottom-up adaptation mechanisms to automatically insert fillers and adjust motion shape on the basis of changes in the prediction of time events of interlocutor behavior it synchronizes to. Thus, the combination of Elckerlyc's synchronization to predicted interlocutor events with ACE's bottom-up last minute shape adaptation thus allows us to address an even wider range of interactional coordination scenarios.

References

1. Kendon, A.: Gesticulation and speech: Two aspects of the process of utterance. In Key, M.R., ed.: The relation of verbal and nonverbal communication. Mouton (1980) 207– 227
2. McNeill, D.: Hand and Mind: What Gestures Reveal about Thought. University of Chicago Press (1995)

[5] Goodwin describes another example of such adjustments in conversation: when a listener utters an *assessment* feedback, the speaker, upon recognizing this, will slightly delay subsequent speech (e.g. by an inhalation or production of a filler) until the listener has completed his assessment [19].

3. Bernieri, F.J., Rosenthal, R.: Interpersonal coordination: Behavior matching and interactional synchrony. In Feldman, R.S., Rimé, B., eds.: Fundamentals of Nonverbal Behavior. Studies in Emotional and Social Interaction. Cambridge University Press (1991)
4. Kopp, S., Wachsmuth, I.: Synthesizing multimodal utterances for conversational agents. Computer Animation and Virtual Worlds **15**(1) (2004) 39– 52
5. Reidsma, D., van Welbergen, H., Zwiers, J.: Multimodal plan representation for adaptable BML scheduling. In: Intelligent Virtual Agents. Volume 6895 of LNCS., Springer Berlin / Heidelberg (2011) 296–308
6. Salem, M., Kopp, S., Wachsmuth, I., Joublin, F.: Towards an integrated model of speech and gesture production for multi-modal robot behavior. In: Symposium on Robot and Human Interactive Communication. (2010) 649 – 654
7. Schlangen, D., Skantze, G.: A general, abstract model of incremental dialogue processing. Dialogue & Discourse **2**(1) (2011) 83–111
8. Nijholt, A., Reidsma, D., van Welbergen, H., op den Akker, H., Ruttkay, Z.M.: Mutually coordinated anticipatory multimodal interaction. In: Verbal and Nonverbal Features of Human-Human and Human-Machine Interaction. Volume 5042 of LNCS., Springer (2008) 70– 89
9. Kopp, S., Krenn, B., Marsella, S.C., Marshall, A.N., Pelachaud, C., Pirker, H., Thórisson, K.R., Vilhjálmsson, H.H.: Towards a common framework for multimodal generation: The Behavior Markup Language. In: Intelligent Virtual Agents. Volume 4133 of LNCS., Springer (2006) 205– 217
10. Thiebaux, M., Marshall, A.N., Marsella, S.C., Kallmann, M.: Smartbody: Behavior realization for embodied conversational agents. In: Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems (2008) 151– 158
11. Heloir, A., Kipp, M.: Real-time animation of interactive agents: Specification and realization. Applied Artificial Intelligence **24**(6) (2010) 510–529
12. Čereković, A., Pandžić, I.S.: Multimodal behavior realization for embodied conversational agents. Multimedia Tools and Applications (2010) 1–22
13. van Welbergen, H., Reidsma, D., Ruttkay, Z.M., Zwiers, J.: Elckerlyc: A BML realizer for continuous, multimodal interaction with a virtual human. Journal on Multimodal User Interfaces **3**(4) (2010) 271– 284
14. Mancini, M., Niewiadomski, R., Bevacqua, E., Pelachaud, C.: Greta: a SAIBA compliant ECA system. In: Troisiéme Workshop sur les Agents Conversationnels Animés. (2008)
15. van Welbergen, H., Xu, Y., Thiebaux, M., Feng, W.W., Fu, J., Reidsma, D., Shapiro, A.: Demonstrating and testing the BML compliance of BML realizers. In: Intelligent Virtual Agents. Volume 6895 of LNCS., Springer Verlag (2011) 269–281
16. van Welbergen, H., Zwiers, J., Ruttkay, Z.M.: Real-time animation using a mix of physical simulation and kinematics. Journal of Graphics, GPU, and Game Tools **14**(4) (2009) 1– 21
17. Reidsma, D., de Kok, I., Neiberg, D., Pammi, S., van Straalen, B., Truong, K.P., van Welbergen, H.: Continuous interaction with a virtual human. Journal on Multimodal User Interfaces **4**(2) (2011) 97– 118
18. Reidsma, D., Dehling, E., van Welbergen, H., Zwiers, J., Nijholt, A.: Leading and following with a virtual trainer. In: Workshop on Whole Body Interaction, University of Liverpool (2011)
19. Goodwin, C.: Between and within: Alternative sequential treatments of continuers and assessments. Human Studies **9**(2-3) (1986) 205– 217