Multimodal Plan Representation for Adaptable BML Scheduling

Dennis Reidsma, Herwin van Welbergen, and Job Zwiers

Human Media Interaction, University of Twente
{h.vanwelbergen,d.reidsma,j.zwiers}@utwente.nl

Abstract. In this paper we show how behavior scheduling for Virtual Humans can be viewed as a constraint optimization problem, and how Elckerlyc uses this view to maintain a extensible behavior plan representation that allows one to make micro-adjustments to behaviors while keeping constraints between them intact. These capabilities make it possible to implement tight mutual behavior coordination between a Virtual Human and a user, without requiring to re-schedule every time an adjustment needs to be made.

This paper describes a novel intermediate plan represention for the multimodal behavior of a Virtual Human (VH). A VH displays the verbal and nonverbal behavior that has been specified by higher level communicative intent and behavior planning processes in the context of a dialog or other interaction. Our aim is to achieve capabilities for human-like interpersonal coordination. For this, we need to be able to make on-the-fly adjustments to the behavior being displayed [7,9]. Goodwin describes an example of such adjustments: when a listener utters an *assesment* 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 [3]. As another example, when a virtual sports coach is performing an exercise along with the user, it needs to continually update the exact timing with which it performs the movements in order to stay synchronized with the user. We have designed our flexible multimodal plan representation specifically for enabling these, and other, on-the-fly adjustments.

The behavior plan representation forms an intermediate level between the scheduling of specified behavior and the surface realization on the embodiment of the VH. The behavior is specified using the Behavior Markup Language (BML) [8], defining the form of the behavior and the constraints on its timing (see Fig. 1). A BML Realizer is responsible for executing the behaviors in such a way that the time constraints specified in the BML blocks are satisfied. Realizer implementations typically handle this by separating the BML scheduling process from the playback process (see also [5]). The scheduling process converts the BML blocks to a multimodal behavior plan that can be directly displayed by the playback process on the embodiment of the VH (see Fig. 2). The flexibility of our plan representation, used in our BML realizer Elckerlyc [13] makes it possible to make on-the-fly adjustments to the plan, during playback, while keeping the constraints on the behaviors intact.

H. Högni Vilhjálmsson et al. (Eds.): IVA 2011, LNAI 6895, pp. 296–308, 2011.

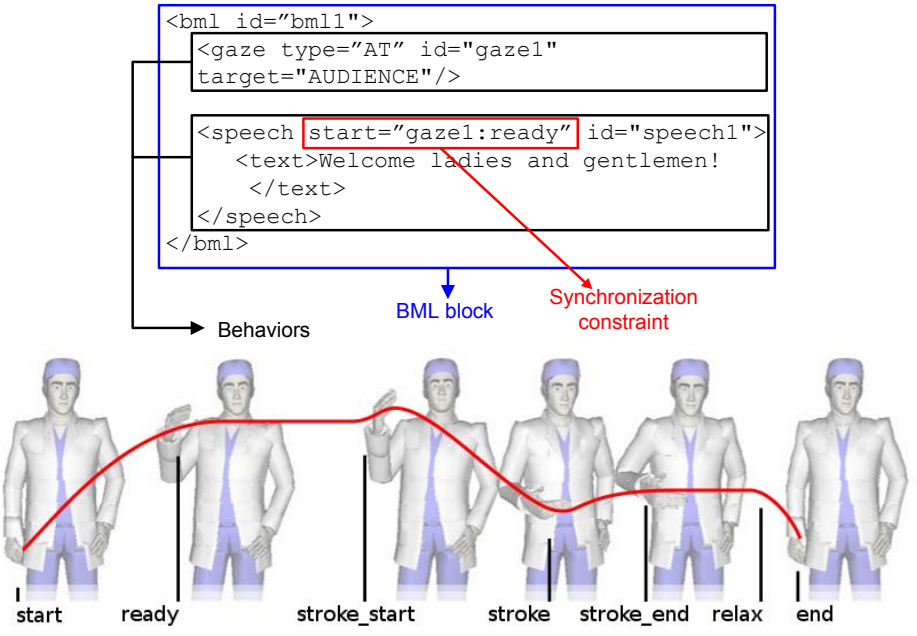


Fig. 1. An example BML script, and the standard phases of a BML behavior; lowermost picture from `http://wiki.mindmakers.org/projects:bml:main`

Our approach requires that we view the scheduling as a constraint optimization problem. Therefore, we will start the paper with an explanation of scheduling in these terms. Both standard constraint optimization techniques and custom BML scheduling algorithms haven been used to solve it. In most BML Realizers scheduling the stream of BML blocks results in a *rigid* multimodal realization plan. Once scheduled, the plan cannot be modified very well – at best, a Realizer allows one to drop (a subset of) the current plan and replace it with a new plan. The more flexible plan representation used in Elckerlyc allows use to interrupt behaviors, change the timing of synchronisation points, add additional behaviors, and change the parameterization of behaviors on-the-fly while keeping the constraints intact. This makes it eminently suitable for VH applications in which a tight mutual coordination between user and VH is required.

1 BML Scheduling as a Constraint Problem

Fig. 2 shows how the scheduling process creates and maintains the intermediate multimodal behavior plan that will be displayed on the VH's embodiment at playback time. A new BML block $u \in \mathbf{u}$ (the collection of all blocks) is sent to the scheduler at time ct (indicated by the vertical white bar). The block u specifies new behaviors $\mathbf{b}$ with sync points $\mathbf{s}$ (such as start, stroke, or end) and their alignment. The scheduling process of a Realizer updates the current

multimodal behavior plan on the basis of u. The behaviors are added to the plan subject to a set of timing constraints $\mathbf{c}$. Firstly, there are the constraints that are explicitly defined in the BML block specification. Secondly, there are certain implicit constraints that hold for any BML block (e.g., behaviors should start before they end). Thirdly, a specific Realizer can impose additional constraints upon the scheduling, for example motivated by biological capabilities of humans. Finally, Block Level Constraints, as specified by the scheduling attribute in the BML block, define the relation between the start of the to-be-scheduled BML block and the behaviors already present in the current behavior plan (see the difference between the two examples in Fig. 2). The four types of constraints are described in more detail below; due to a lack of space, we refer the reader to [12] for the formal definitions and equations.

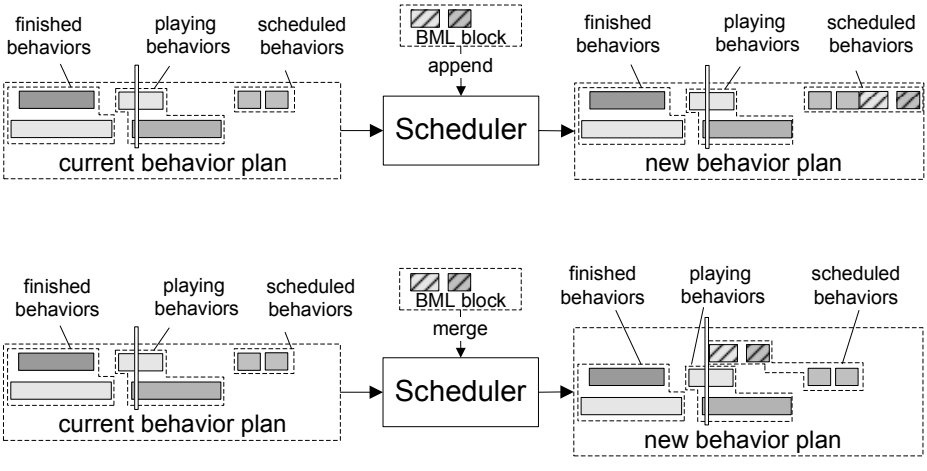


Fig. 2. The scheduling process. The white bar indicates the current time. The new BML block defines how the currently playing and planned behaviors are updated and which new behaviors are inserted, using a scheduling attribute. `append` (top) indicates that the behaviors in the BML block are to be inserted after all behaviors in the current plan. `merge` (bottom) specifies that the behaviors in the BML block are to be started at the current time.

A scheduling function $f : \mathbf{s} \rightarrow \mathbf{t}$ maps sync points s to global time t. The function *blockstart* : $\mathbf{u} \rightarrow \mathbf{t}$ maps blocks u to their global start time t. The goal of scheduling is to find a mapping $\mathbf{f}$ for all sync points in all behaviors in the block and to determine the start time of the block, in such a way that all constraints are satisfied. Furthermore, a Realizer may specify an additional cost function to select the 'best' solution from the possible solutions, e.g. in terms of amount of stretching of animations, acceleration of body parts, etcetera.

1.1 Explicit Constraints

Explicit time constraints are specified in the BML expression, as a relation between *sync references*. A sync reference consists of either a time value in seconds,

denoting an offset from the start of the BML block, or a sync point of one of the behaviors (possibly with an offset). For example, the behavior definition <gaze id="g1" ready="beh1:start+1"> in BML block $bml1$ defines a constraint between the sync refs $[bml1 : g1 : ready + 0]$ and $[bml1 : beh1 : start + 1]$.

A time constraint expresses a time relation between two or more sync references. BML defines two types of time relations:

- *before/after*: sync ref a occurs before (or after) sync ref b.
- *at*: sync refs a and b occur at the same time.

1.2 Implicit Constraints

Apart from the explicit constraints defined in the BML block, several implicit constraints act upon f:

1. Sync points may not occur before the block in which they occur is started.
2. Behaviors should have a nonzero duration.
3. The default BML sync points of each behavior (`start`, `ready`, `stroke_start`, `stroke`, `stroke_end`, `relax`, `end`) must stay in that order.

1.3 Realizer Specific Constraints

Realizers might impose additional constraints, that are typically behavior specific. They may, for example, be due to a technical limitation. Most Text-To-Speech systems do not allow one to make detailed changes to the timing of the generated speech. Therefore, Realizers typically forbid scheduling solutions that require the stretching of speech behaviors beyond the default timing provided by the TTS system. Constraints may also be theoretically motivated. A Realizer might, e.g., forbid solutions that require a VH to gesture at speeds beyond its physical ability.

1.4 Block Level Constraints

The scheduling attribute associated with a BML Block (see also Fig. 2) defines constraints on the start of the block in relation to the set of current behaviors in the block and the current global time ct. BML defines the following scheduling attributes:

1. `merge`: start the block at ct.
2. `replace`: remove all behaviors from the current plan, start the block at ct.
3. `append`: start the block as soon as possible after all behaviors in the current plan have finished (but not earlier than ct).

2 Existing BML Scheduling Solutions

In this section we describe the scheduling solutions implemented in the BML Realizers SmartBody and EMBR. Both Realizers implement an additional constraint that is not (yet) officially defined in the BML standard: each behavior, and each block, are supposed to start as early as possible, as long as they satisfy all other constraints.

2.1 SmartBody Scheduling

SmartBody's [11] scheduling algorithm assigns an absolute timing for sync points in each behavior by processing behaviors in the order in which they occur within a BML block. Each behavior is scheduled to adhere to its BML block timing constraint and to the timing constraints posed by their predecessors in the BML block. If two time constraints on a behavior require certain phases of that behavior to be stretched or skewed, the scheduler achieves this by stretching or skewing the behavior uniformally, to avoid discontinuities in animation speed. Their scheduling mechanism can result in some time constraints being scheduled into the past (that is, before the start of the BML block). A final normalization pass shifts, where needed, connected clusters of behaviors forward in time to fix this. SmartBody cannot handle before/after constraints yet, but does comply with all explicit constraints and implicit constraints that do not concern before and after constraints.

Because the SmartBody scheduling algorithm schedules behaviors in BML order, the result is dependent on the order in which the behaviors are specified in the BML block. At worst, this may lead to situations in which BML cannot be scheduled in one order while it can be in another. For example, the BML block in Example 1(a) cannot be scheduled because the timing of the `nod1` is determined first, and the scheduler attempts to retime `speech1` to adhere to this timing. Most speech synthesis systems, including the one used in SmartBody, disallow such retiming. If the behavior order is changed, as in Example 1(b), then `speech1` is scheduled first, and `nod1` will adhere to the timing imposed by `speech1`.

That being said, the SmartBody scheduling algorithm is easy to implement and provides rapid scheduling. In practice, most BML scripts are simple and the SmartBody scheduler will find a reasonable scheduling solution for such scripts.

2.2 EMBR

EMBR [4,6] uses a constraint optimization technique to solve the scheduling problem. The EMBR scheduler first solves the absolute value of all BML sync points in speech. A timing constraint solver then solves for the timing of the remaining nonverbal behaviors. Synchronization constraints might require the stretching or skewing of behavior phases as compared to the defaults given in the behavior lexicon. The constraint solver uses the sum of 'errors' (in seconds) of the stretch or skew over all behaviors as its cost function. It thus finds solutions in which the overall stretch/skew is minimized. The EMBR scheduler can schedule BML blocks containing before and after constraints, and favors solutions that result in more natural behavior (for EMBR's measure of the naturalness: minimal overall behavior stretching/skewing).

3 Scheduling and Plan Representation in Elckerlyc

Elckerlyc is a BML Realizer designed specifically for continuous interaction [13] Its multimodal behavior plan can continually be updated: the timing of certain

BML Example 1. Two BML scripts demonstrating SmartBody's order dependent scheduling solution.

(a) BML script that cannot be scheduled using the SmartBody scheduling algorithm.

```
<bml id="bml1">
  <head id="nod1" action="ROTATION" rotation="NOD" start="speech1:start"
   end="speech1:sync1"/>
  <speech id="speech1">
    <text>Yes,<sync id="sync1"> that was great.</text>
  </speech>
</bml>
```

(b) BML script that can be scheduled using the SmartBody scheduling algorithm.

```
<bml id="bml1">
  <speech id="speech1">
    <text>Yes,<sync id="sync1"> that was great.</text>
  </speech>
  <head id="nod1" action="ROTATION" rotation="NOD"
    start="speech1:start" end="speech1:sync1"/>
</bml>
```

synchronisation points can be adjusted, ongoing behaviors can be interrupted using interrupt behaviors, behaviors can be added, and the parametrisation of ongoing behaviors can be changed at playback time.

In order to achieve this flexibility, Elckerlyc needs not only to be able to schedule BML specifications into a multimodal behavior plan that determines the surface realization of the behaviors, but also needs to maintain information about how these surface realizations relate to the original BML specification. This allows the scheduler to figure out which surface realizations need to be changed, when changes to BML behaviors are requested.[1]

Elckerlyc's flexible plan representation allows these modifications, while keeping the constraints on the behavior plan consistent. In this section we describe this plan representation, and the architecture of the scheduling component in Elckerlyc.

3.1 Additional Behavior Plan Constraints in Elckerlyc

Elckerlyc allows a number of additional constraints of different types. The most important ones are described here.

Implicit Constraints: Whitespace. Similar to SmartBody and EMBR, Elckerlyc adds a set of constraints to enforce that there is no 'unnecessary whitespace'

[1] The mechanisms for specifying these changes are described in [ANON].

between behaviors. That is, each behavior, as well as each block, is supposed to start as early as possible, as long as it satisfies all other constraints.

Block Level Constraint: Scheduling Types. In addition to the `merge` and `append` scheduling attributes, Elckerlyc provides the `append-after(X)` attribute. Append-after starts a BML block directly after a selected set of behaviors in the current behavior plan (those from all blocks in **X**) are finished.

3.2 Elckerlyc's Plan Representation

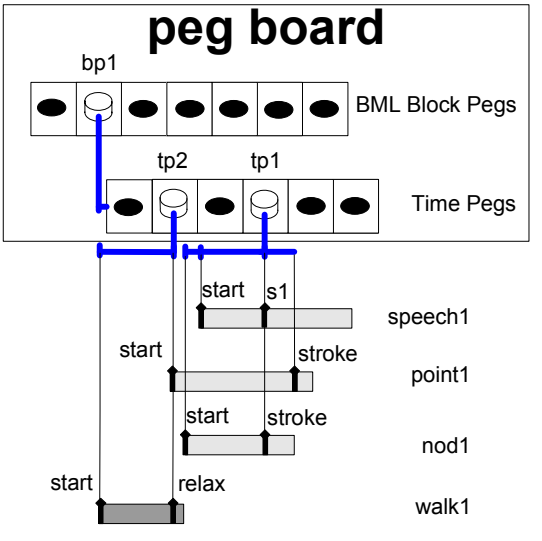
Central to Elckerlyc's plan representation is the Peg Board. The sync points of each behavior in the multimodal plan are associated with Time Pegs on the Peg Board. These Time Pegs can be moved, changing the timing of the associated sync points. If two sync points are connected by an 'at' constraint, they share the same Time Peg. This Time Peg can then be moved without violating the 'at' constraint, because this simultaneously changes the actual time of both sync points. Time Pegs provide local timing (that is, as offset from the start of the block), being connected to a BML Block Peg that provides representation of the start time of the corresponding BML block. If the BML Block Peg is moved, all Time Pegs associated with it move along. This allows one to move the block as a whole, keeping the intra-block constraints consistent (see Fig. 3).

A dedicated BML Block management state machine automatically updates the timing of the BML Block Pegs in reaction to behavior plan modifications that occur at runtime, to maintain the BML Block constraints. For example, when a block b_i was scheduled to occur immediately after all behaviors already present in the plan, and the immediately preceding behaviors in the plan are removed from the plan through an `interrupt` behavior, the state machine will automatically move the BML Block Peg of b_i to close the resulting gap.

3.3 Resolving Constraints to Time Pegs

Relative 'at' synchronization constraints that share a sync point (behavior id, sync id pair) should be connected to the same Time Peg. Such 'at' constraints may involve a fixed, nonzero timing offset, for example when a nod is constrained to occur exactly half a second after the stroke of a gesture. Such offsets are maintained by Offset Pegs. An Offset Peg is a Time Peg that is restrained to stay at a fixed offset to its linked Time Peg. If the Offset Peg is moved, its linked Time Peg moves with it and vice-versa. Offset Pegs can also be added by the scheduler for other reasons. For example, if the start sync is not constrained in a behavior, it may be resolved as an Offset Peg. That is: the start sync of the behavior is linked to the closest Time Peg of another sync point within the behavior. If this other Time Peg is moved, the start of the behavior is moved with it. If a behavior is completely unconstrained, a new Time Peg is created and connected to its start sync. BML Example 2 shows how Time Pegs are resolved for an example BML constraint specification.

BML Example 2. Resolving a BML constraint specification to a Time Pegs specification. A Time Peg $tp1$ connects relative 'at' constraints $[[bml1 : speech1 : s1 + 0], [bml1 : nod1 : stroke + 0]]$, and $[[bml1 : speech1 : s1 + 0], [bml1 : point1 : stroke + 0.5]]$. Another Time Peg $tp2$ is created for the 'at' constraint $[[bml1 : point1 : start + 0], [bml1 : walk1 : relax + 0]]$. Since the start time of speech1, nod1, and walk1 is not constrained, they are attached to an Offset Pet linked to the closest other Time Peg in the respective behaviors. The BML Block itself (with id `bml1`) is connected to BML Block Peg `bp1`. All Time Pegs are connected to this Block Peg.




```
<bml id="bml1">
    <speech id="speech1">
      <text>As you can see on <sync id="s1"> this painting, ...</text>
    </speech>
    <gesture id="point1" start="walk1:relax" type="POINT"
    target="painting1" stroke="speech1:s1+0.5"/>
    <head id="nod1" action="ROTATION" rotation="NOD" stroke="speech1:s1"/>
    <locomotion id="walk1" target="painting1"/>
</bml>
```

Each BML Block has its own associated BML Block Peg that defines its global start time. Time Pegs are linked to their associated BML Block Peg. Some behaviors have constraints (and thus Time Pegs) that are linked to external global Pegs, used to synchronize behavior with external events. These are hooked up to a special, unmovable global BML Block Peg at global $t = 0$. See Fig. 3 for a graphical representation of these relations. The actual time of a BML Block Peg is first estimated to be *ct* (the time at which it is being scheduled). When playback of the Block Peg is started, its time is updated to reflect its actual start time. Since the Time Pegs inside the block are attached to the Block Peg, they will now also adhere to the actual start time of the block.

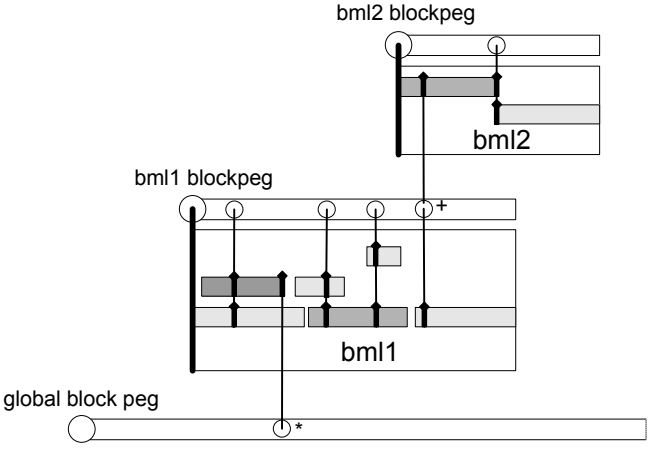


Fig. 3. Each BML block has its associated BML Block Peg. Internal constraints are linked to Time Pegs associated with this BML Block Peg. BML block `bml1` contains a constraint that is linked to an external Time Peg (marked with *). BML block `bml2` is block scheduled with the tight-merge scheduling algorithm. It has a constraint whose timing is defined by a Time Peg from BML block `bml1` (marked with +).

3.4 Scheduling in Elckerlyc

In Elckerlyc, scheduling consists of resolving the constraints in a BML block to Time Pegs (see Section 3.3), and assigning the Time Pegs a first prediction of their execution time. Elckerlyc's main scheduling contribution is in its flexible behavior plan representation described in Section 3.3; Elckerlyc currently uses SmartBody's scheduling algorithm to assign time predictions to the Time Pegs. The architecture of Elckerlyc is set up in such a way that this scheduling algorithm can easily be replaced by other algorithms (e.g. a custom constraint solver such as that of EMBR).

Scheduling Architecture. The Elckerlyc scheduling architecture uses an interplay between different unimodal Engines that provide the central scheduler with detailed information on the possible timing of behaviors, given their BML

description and time constraints. Elckerlyc's multimodal plan representation is managed using unimodal plans in each Engine. These unimodal plans contain Timed Plan Units, the timing of which is linked to Time Pegs on the PegBoard. Elckerlyc's Scheduler communicates with these Engines (e.g. Speech Engine, Animation Engine, see also Fig. 4) through their abstract interface (see below). It knows for each BML behavior type which Engine handles it.

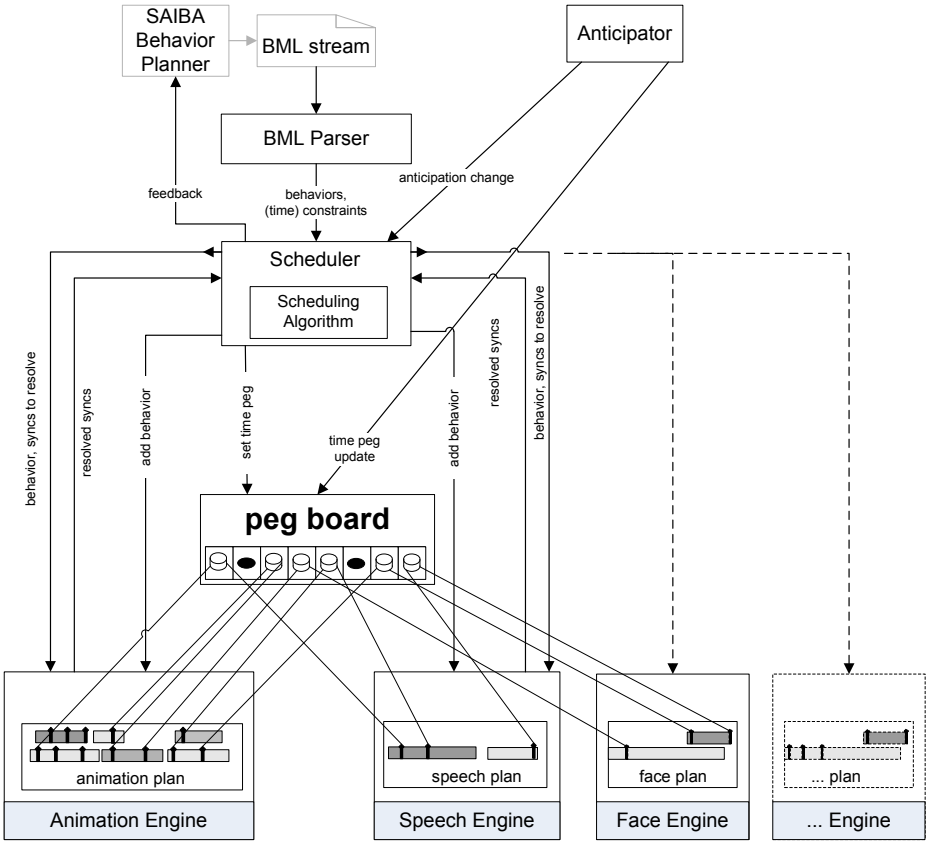


Fig. 4. Elckerlyc's scheduling architecture. The Anticipator is used to adapt the time stamps of Time Pegs based on predictions of the timing of external events, which is useful when the VH should exhibit tight behavior synchronisation with conversation partners.

Interfacing with the Engines Each Engine implements functionality to:

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

Note that an Engine can be queried for time constraints on behavior without adding it to the plan. This allows a scheduler to try out multiple constraint configurations on each behavior before it commits to a specific behavior plan. Also note that all communication with the Engine is in terms of BML behaviors. It is up to the Engine to map the BML behaviors to Timed Plan Units. The validity check is typically used to check if a valid plan is retained after certain TimePegs have been moved. All implemented Engines check if the order of the TimePegs of each behavior is still correct. Each Engine can also add validity checks specific to its output modality.

Scheduling Algorithm. The scheduler delegates the actual scheduling to a dedicated algorithm, using the strategy pattern [2]. The scheduling algorithm assigns a first prediction of the timing of each Time Pegs to them, given the current multimodal behavior plan and a parsed BML block that is to be scheduled. Elckerlyc is designed in such a way that the scheduling algorithm can easily be changed: the BML parsing and block progress management are separated from the scheduling algorithm, and the Engines provide generic interfaces that provide a scheduling algorithm with the timing of unknown constraints on behaviors, given certain known constraints.

Elckerlyc's scheduling algorithm is based on the SmartBody Scheduler described in Section 2.1. The behaviors are processed in the order in which they occur in the BML block. The first behavior in the BML block is constrained only by its absolute time constraints and constraint references to external behaviors. Subsequent behaviors are timed so that they adhere to time constraints imposed by the already processed behaviors (including those of previous blocks). Elckerlyc's BML Parser lists all constraints on each behavior. Our current scheduler delegates resolving these unknown time constraints directly to the Engine that is dedicated to the given behavior type. Subsequently, the behavior, on which all time time constraints are now resolved, is added to its Plan.

3.5 Managing Adjustments of the Behavior Plan during Behavior Playback

Once a BML block is scheduled, several changes can occur to its timing at playback time. Such changes may, for example, be initiated by a Time Peg being moved for external reasons (e.g., to postpone a speech phrase until the interlocutor finished uttering an assessment feedback, as explained in the introduction), or by other behaviors in the plan being removed. Since the sync points of behaviors are symbolically linked to the Time Pegs, timing updates are handled automatically (stretching or shortening the duration of behaviors when required) and the explicit constraints of Section 1.1 remain satisfied. Plan changes, and constraint satisfaction after plan changes, are achieved in an efficient manner, that is, without requiring a time consuming scheduling action for minor plan adjustments. Interrupting a behavior in a BML block might shorten the length of the block. Since the BML Block management state machine dynamically manages the block end, shortening the block whenever this happens, the whitespace and append constraints automatically remain satisfied.

These and other kinds of microadjustmenst to behavior plans have been experimented with in a number of applications. The latest version of the Reactive Virtual Trainer performs fitness exercises along with the user, adjusting the timing of its performance to that of the user [1]. In experiments on Attentive Speaking, a route guide slightly delays its speech to make room for listener responses from the user [10] (using a Wizard of Oz setup for detecting start and end of listener responses). Other applications and scenarios have been described elsewhere; videos and demonstrations may be found on the Elckerlyc web site and in the open source code release.

More significant updates might require re-scheduling of behaviors, such as when a Time Peg, linked to the start of a behavior, is moved to occur *after* the end of the same behavior. To check for such situations, the Scheduler asks each Engine whether its current plan is still valid (i.e., its constraints are still satisfied). 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.

4 Conclusion

We showed in this paper how the BML scheduling process can be viewed as a constraint problem, and how Elckerlyc uses this view to maintain a flexible behavior plan representation that allows one to make micro-adjustments to behaviors while keeping constraints between them intact. In Elckerlyc, scheduling is modeled as an interplay between different unimodal Engines that provide detailed information on the timing of the behaviors that are to be realized. The seperation of concerns between unimodal behavior timing, BML parsing, BML block progress management and multimodal scheduling makes it easy to exchange Elckerlyc's scheduling algorithm by a different one as well as to add new modalities. Thanks to the capability for on-the-fly plan adjustments, Elckerlyc is eminently suitable for Virtual Human applications in which a tight mutual coordination between user and Virtual Human is required.

Acknowledgements. This research has been supported by the GATE project, funded by the Dutch Organization for Scientific Research (NWO).

References

1. Dehling, E.: The Reactive Virtual Trainer. Master's thesis, University of Twente, Enschede, the Netherlands (2011)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
3. Goodwin, C.: Between and within: Alternative sequential treatments of continuers and assessments. Human Studies 9(2-3), 205–217 (1986)

4. Heloir, A., Kipp, M.: Real-time animation of interactive agents: Specification and realization. Applied Artificial Intelligence 24(6), 510–529 (2010)
5. Herman, M., Albus, J.S.: Real-time hierarchical planning for multiple mobile robots. In: Proc. DARPA Knowledge -Based Planning Workshop (Proceedings of the DARPA Knowledge-Based Planning Workshop), pp. 22-1–22-10
6. Kipp, M., Heloir, A., Schröder, M., Gebhard, P.: Realizing multimodal behavior: Closing the gap between behavior planning and embodied agent presentation. In: Proc. IVA, pp. 57–63. Springer, Heidelberg (2010)
7. Kopp, S.: Social resonance and embodied coordination in face-to-face conversation with artificial interlocutors. Speech Communication 52(6), 587–597 (2010), speech and Face-to-Face Communication
8. 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: Gratch, J., Young, M., Aylett, R.S., Ballin, D., Olivier, P. (eds.) IVA 2006. LNCS (LNAI), vol. 4133, pp. 205–217. Springer, Heidelberg (2006)
9. Nijholt, A., Reidsma, D., van Welbergen, H., op den Akker, R., Ruttkay, Z.: Mutually coordinated anticipatory multimodal interaction. In: Esposito, A., Bourbakis, N.G., Avouris, N., Hatzilygeroudis, I. (eds.) HH and HM Interaction. LNCS (LNAI), vol. 5042, pp. 70–89. Springer, Heidelberg (2008)
10. 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 (in press, 2011)
11. Thiebaux, M., Marshall, A.N., Marsella, S.C., Kallmann, M.: Smartbody: Behavior realization for embodied conversational agents. In: Proc. AAMAS, pp. 151–158 (2008)
12. van Welbergen, H.: Specifying, scheduling and realizing multimodal output for continuous interaction with a virtual human. Ph.D. thesis, University of Twente, Enschede, NL (2011)
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), 271–284 (2010)