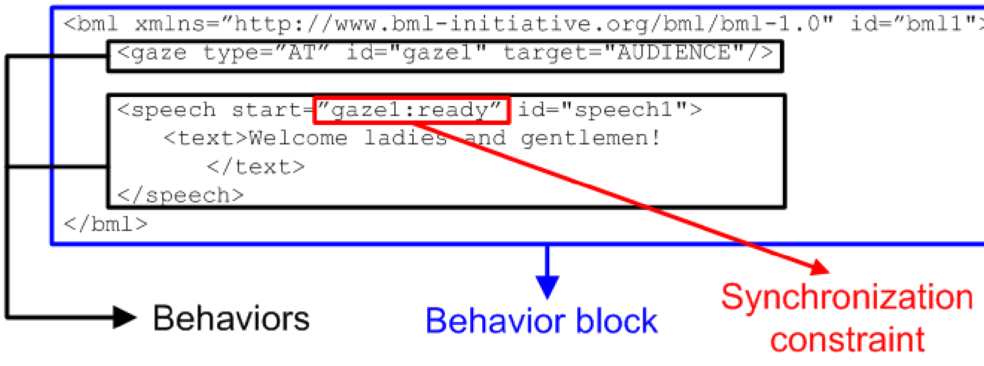
This document introduces and describes version 1.0 of the Behavior Markup Language standard. It contains background information, descriptions of typical use contexts, and, most importantly, the syntactic and semantic details of the XML format of the Behavior Markup Language.

Introduction

The Behavior Markup Language, or BML, is an XML description language for controlling the verbal and nonverbal behavior of (humanoid) embodied conversational agents (ECAs). A BML block (see figure 1) describes the physical realization of behaviors (such as speech and gesture) and the synchronization constraints between these behaviors. BML is not concerned with the communicative intent underlying the requested behaviors. The module that executes behaviors specified in BML on the embodiment of the ECA is called a BML Realizer.



```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0" id="bml1">
  <gaze type="AT" id="gaze1" target="AUDIENCE"/>

  <speech start="gaze1:ready" id="speech1">
     <text>Welcome ladies and gentlemen!
        </text>
  </speech>
</bml>
```

Figure 1: Example of a BML Request

The BML Standard consists of a small and lean core, plus a few clearly defined mechanisms for extending the language.

Lean Core

Full Name	BML Core Standard
Status	Required
XML Namespace	http://www.bml-initiative.org/bml/bml-1.0
Examples	basic speech, pointing gestures

The Core of the BML Standard defines the form and use of BML blocks, mechanisms for synchronization, the basic rules for feedback about the processing of BML messages (see later in this document), plus a number of generic basic behaviors. BML compliant realizers implement the complete BML Core Standard and provide a meaningful execution for all its behavior elements. Some realizers might offer only partial compliance, for example because they only steer a head (and therefore do not need to interpret bodily behaviors). In that case, a realizer should at least provide an exception/warning feedback when being requested to execute unsupported Core Standard behaviors (see Feedback).

Core Extensions

Full Name	BML Core Extensions
Status	Optional, but if a realizer implements the functionality of a Core Extension, it should exactly follow the standard specification.
XML Namespace	http://www.bml-initiative.org/bml/... (last part is specified in the definition of the Core Extension)
Examples	FACS face expressions, SSML description extension for speech

BML provides several standardized mechanisms for extension. One can define new behaviors (in a custom namespace), or extend upon Core behaviors by adding custom attributes. Description extensions provide a standardized manner for a user to give more detail about how the BML Realizer should realize a given instance of a core behavior, while allowing a fallback to the Core specification when the BML Realizer does not support the extension.

The BML standard defines a number of Core Extensions, both in the form of additional behaviors and in the form of description extensions. The Core Extensions provide behaviors and description extensions that we do not want to make mandatory, but we do want to be implemented in a standardized way whenever a BML Realizer implements them. We encourage authors of realizers to collaborate and define shared behavior types and descriptions beyond those provided by the core extensions.

Global Context

SAIBA

The Behavior Markup Language is part of the SAIBA multi-modal Behavior Generation Framework (see Figure 2). In this framework, the intention for the ECA to express something arises in the Intent Planner. The Behavior Planner is responsible for deciding which multi-modal behaviors to choose for expressing the communicative intent (through speech, face expressions, gestures, etc) and for specifying proper synchronization between the various modalities. This multi-modal behavior is specified in the form of BML messages. A BML Realizer is responsible for physically realizing the specified BML message through sound and motion (animation, robot movement, ...), in such a way that the time constraints specified in the BML block are satisfied. At runtime, the BML realizer sends back feedback messages to keep the planning modules updated about the progress and result of the realization of previously sent BML messages, allowing, e.g., for monitoring and possible error recovery.

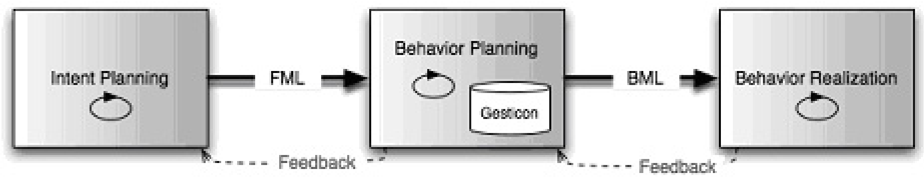


Figure 2: SAIBA Framework

The exact nature of the intent and behavior planning processes is left unspecified here. As far as the BML Realizer is concerned, it makes no difference whether BML messages are the result of a complicated multi-modal affective dialog system, or are simply predefined BML messages pulled from a library of pre-authored materials.

BML Messaging Architecture

BML does not prescribe a specific message transport. Different architectures have drastically different notions of a message. A message may come in the form of a string, an XML document or DOM, a message object, or just a function call. However, no matter what message transport is used, the transport and routing layer should adhere to the following requirements:

- Messages must be received in sent order.
- Messages must contain specific contents that can be fully expressed as XML expressions in the format detailed in this document.

Currently, there are two types of messages:

BML Requests

- Sent by the Behavior Planner to the Behavior Realizer.
- BML requests are sent as `<bml>` blocks containing a number of behavior elements with synchronization.

Feedback Messages

- Sent by the Behavior Realizer.
- Used to inform the planner (and possibly other processes) of the progress of the realization process.

The BML Realizer

Conceptually, BML Realizers execute a multi-modal plan that is incrementally constructed (scheduled) on the basis of a stream of incoming BML Requests (see Figure 3)). A BML Realizer is responsible for executing the behaviors specified in each BML request sent to it, in such a way that the time constraints specified in the BML request are satisfied. If a new request is sent before the realization of previous requests has been completed, a composition attribute determines how to combine the behaviors in the new request with the behaviors from earlier requests (see documentation of composition attribute).
Each BML Request represents a scheduling boundary. That is: if behaviors are in the same BML request, this means that the constraints between them are resolved before any of the behaviors in the request is executed.

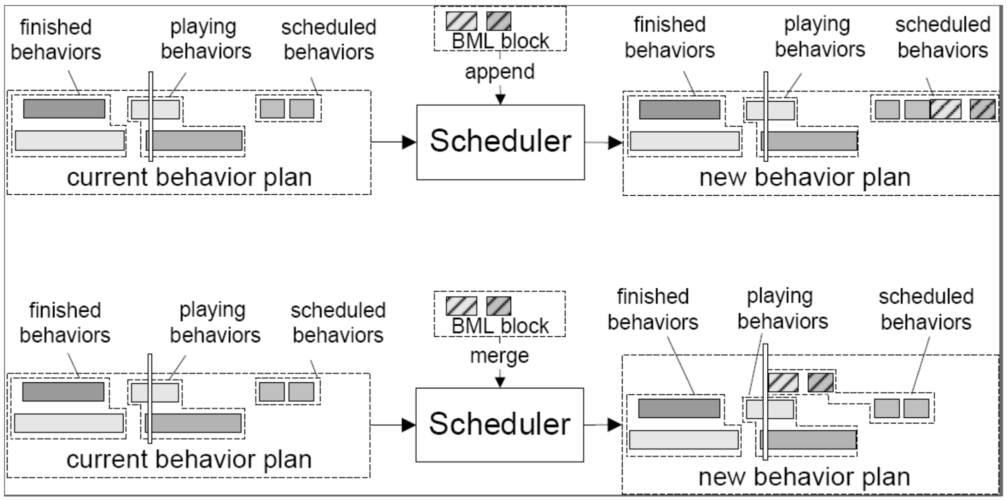


Figure 3: Dealing with an incoming stream of BML Requests

The State of an ECA

BML assumes that there is something like a ground state of the ECA (Embodied Conversational Agent). This state comprises several elements, such as the permanent posture or the ground state of the face. For example, when a temporary `<<posture>>` behavior ends, the ECA reverts to the posture defined in the ground state; when a temporary face expression ends, the face of the ECA reverts to a ground state. Some types of behavior have a residual effect. That is, after the end time of the behavior has been reached, the ground state of the ECA will be different than before the behavior started. Such behaviors are generally names `<<...Shift>>`. Details can be found at the documentation of each particular element; here we present a table of dimensions to the ground state of the ECA, and behaviors that may influence this ground state.

Ground state aspect	**Behaviors that change this state**
Body posture	`<postureShift>`
Head pose	`<headDirectionShift>`
Face expression	`<faceShift>`
Gaze direction	`<gazeShift>`
Location in the world	`<locomotion>`

XML Format: Values and Types

Before describing the various XML elements in the BML Standard, we describe here the available attribute types.
We use camelCase throughout for element names and attribute names. Values of type openSetItem and closedSetItem defined in this document are generally all uppercase. The names of default syncpoints for various behavior types are also written in camelCase (e.g., strokeStart).

Attribute Value Types

Values for various types of behavior attributes can be one of the following:

Type	**Description**
ID	An identifier that is unique within a specified context (see `<bml>`and "behavior element"). Adheres to standard XML type ID
`syncref`	Describes the relative timing of sync points (see the section on synchronisation)
`worldObjectID`	Uunique ID of an object in the character's world. Adheres to standard XML type ID
`targetID`	Unique ID referring to a target in the character's world. Adheres to standard XML type ID
`closedSetItem`	A string member from a closed set of strings, where the standard will provide the exhaustive list of strings in the set.
`openSetItem`	A string member from an open set of strings, where the standard may provide a few common strings in the set.
`bool`	Boolean value, either `true` or `false`
`int`	Integer
`float`	Decimal number
`angle`	Decimal number signifying angle in degrees counterclockwise, between (-180, 180].
`string`	An arbitrary string
`direction`	A particular closedSetItem type from the ClosedSet [LEFT, RIGHT, UP, DOWN, FRONT, BACK, UPRIGHT, UPLEFT, DOWNLEFT, DOWNRIGHT]
`vector`	a string of format `"float; float; float"` indicating the x, y, and z coordinates of a vector

Coordinate System and Units

While we prefer specifying behavior by common verbs and nouns, for some attributes or applications it is unavoidable to use precise vectors.

- All units are metric (kilograms, meters, seconds).
- BML assumes a **global** coordinate system in which the positive Y-axis is up. The **local** (character-based) coordinate system1 adheres to the guidelines of the H-Anim standard (v1.1 and H-Anim): *"The humanoid shall be modeled in a standing position, facing in the +Z direction with +Y up and +X to the humanoid's left. The local character-based origin (0, 0, 0) shall be located at ground level, between the humanoid's feet."*

Currently, there are no expressions in BML 1.0 that actually use the local character based coordinate system. However, future versions may introduce references such as "2 meters to the left of the character".

BML Request

<bml>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<bml>`
Attributes	characterId, id, composition
Contents	behaviors of various types, `<required>` blocks, `<constraint>` blocks

All BML behaviors must belong to a `<bml>` behavior block. A `<bml>` block is formed by placing one or more BML behavior elements inside a top-level `<bml>` element. Unless synchronization is specified (see the section on synchronization), it is assumed that all behaviors in a `<bml>` block start at the same time after arriving at the BML realizer.

Syntax

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
     id="bml1" characterId="Alice" composition="MERGE">
 </bml>
```

Attributes

Attribute	Type	Use	Default	Description
`id`	`ID`	required		Unique ID that allows referencing to a particular `<bml>` block. The id 'bml' is reserved.
`characterId`	`worldObjectID`	optional	`""`	A reference towards the controlled character
`composition`	`openSetItem`	optional	MERGE	One among [MERGE, APPEND, REPLACE], defines the composition policy to apply if the current `<bml>` block overlaps with previous such blocks (see below).

Semantics

No Communicative Meaning

The BML specification does not prescribe a communicative meaning for the BML Request. This allows users of BML to specify short spurts of behavior (for example: speech clauses or individual gaze shifts) and generate performances incrementally, or, if they prefer, to construct elaborate performances as a whole and send them in a single request (for example: entire monologues).

Ordering is not meaningful

The order of elements inside the `<bml>` block does not have any semantic meaning. Authors writing BML expressions should not rely on a BML Realizer realizing something in a certain order because it is in a certain order in the BML block

Start time, end time, delays

Each `<bml>` request represents a scheduling boundary. That is: if behaviors are in the same `<bml>` request, this means that the constraints between them are resolved before any of the behaviors in the request is executed.

start time — the start time of a block b is the global timestamp when it actually starts being executed. The start time may be influenced by various delays, as well as by the composition attribute (both explained further below).
end time — the end time of a block is the global timestamp when all behaviors in the block have ended.

When a planner sends a `<bml>` request to a realizer, there will be a slight (hopefully negligible) delay before the behavior actually starts being performed on the embodiment. The transport and routing layer supporting the transmission of a sequence of `<bml>`blocks will introduce a transmission delay; parsing the request and solving the constraints may introduce another delay.

Composition

If a new request is sent before the realization of previous requests has been completed, a composition attribute determines how to combine the behaviors in the new `<bml>` block with the behaviors from prior `<bml>` blocks. The values for the composition attribute have the following meaning.

- **MERGE (default)** — The start time of the new `<bml>` block will be as soon as possible. The behaviors specified in the new `<bml>` block will be realized together with the behaviors specified in prior `<bml>` blocks. In case of conflict, behaviors in the newly merged `<bml>`block cannot modify behaviors defined by prior `<bml>` blocks.
- **APPEND** — The start time of the new block will be as soon as possible after the end time of all prior blocks.
- **REPLACE** — The start time of the new block will be as soon as possible. The new block will completely replace all prior `<bml>` blocks. All behavior specified in earlier blocks will be ended and the ECA will revert to a neutral state before the new block starts.

As an example of a merge conflict, one might consider two consecutive `<bml>` blocks that both specify a right handed gesture, with the timing being such that they should be performed at the same time. When this turns out to be impossible, the gesture in the block that arrived last should be dropped, and an appropriate warning should be issued (see Feedback section)

`<required>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<required>`
Attributes	none
Contents	behaviors of various types, `<constraint>` blocks

It is generally assumed that the behavior realizer will attempt to realize all behaviors in a block, but if some of the behaviors don't successfully complete for some reason, other behaviors still get carried out (see Feedback and Failure and Fallback).

If there is an all-or-nothing requirement for all or some of the behaviors, they can be enclosed in a `<required>` block inside the `<bml>` block.

```
<bml id="bml1" xmlns="http://www.bml-initiative.org/bml/bml-1.0" characterId="Alice">
   <required>
      <gaze id="gaze1" target="PERSON1"/>
      <speech id="speech1"><text>Welcome to my humble abode</text></speech>
   </required>
   <head id="nod1" type="NOD"/>
</bml>
```

Semantics

If behaviors or constraints enclosed in a `<required>` block cannot be realized, the complete `<bml>`block of which the `<required>` block is a part should be aborted, with appropriate feedback.

In the following example, the entire performance in the `<bml>`block will be aborted if either the gaze or the speech behavior is unsuccessful (and an appropriate feedback message sent back from the behavior realizer, see Feedback section), but if only the head nod is unsuccessful, the rest will be carried out regardless (and an appropriate feedback message sent back from the behavior realizer).

Behaviors (Common Aspects)

A behavior element describes one kind of a behavior to the behavior realizer. In its simplest form, a behavior element is a single XML tag with a few key attributes.

```
<bml id="bml1" xmlns="http://www.bml-initiative.org/bml/bml-1.0" character="Alice">
   <gaze id="gaze1" target="PERSON1"/>
</bml>
```

Syntax

This document specifies a number of XML elements for specifying various sorts of behavior. Any behavior element has at least the following attributes:

Attribute	**Type**	**Use**	**Default**	**Description**
`id`	ID	required	`""`	Unique ID that allows referencing to a particular behavior. The id 'bml' is reserved.
`start`	syncref	optional		Determines the start time of the behavior, either as offset relative to the start time of the enclosing `<bml>` block, or relative to another behavior contained in this block or in another block. If no `syncref` are specified for this behavior, start time is 0; if start is unspecified but other syncrefs are given for this behavior, start is determined by the other `syncref` (and the possible duration for this behavior).
`end`	`syncref`	optional	MERGE	local end time of the behavior, either as offset relative to the start time of the enclosing `<bml>` block, or relative to another behavior contained in this block or in another block. If unspecified, the end time will be dependent on the start time, other `syncrefs` specified on this behavior, and the possible duration of the behavior.

In addition, there may be sync attributes concerning other default sync points for a specific behavior type.

Semantics

There are a few aspects concerning the semantics of behaviors that are common to all behavior types.

Timing and Synchronization

Unless synchronization or timing constraints are specified, it is assumed that all behaviors in a `<bml>` block start at the start time of the `<bml>` block. In the section on synchronization, more detail is given concerning how to specify such constraints.

Targets in the world

Some of the behavior types specified in this document, require reference to a **target** in the world (gaze target, point target, ...). A BML Realizer may assume a number of predefined targets, referenced by an attribute value of type worldObjectID.

For next version, we are working on working out a "target" element that allows more control over specification and modification of targets in the world.

Behaviors with residual effect

Some types of behavior have a residual effect. That is, after the end time of the behavior has been reached, the ground state of the ECA will be different than before the behavior started.

An example of a behavior type with a residual effect is `<locomotion>`: after a `<locomotion>` behavior has been completed, part of the ground state of the ECA (in this case: location and orientation in the world) will be different than before, and other behaviors will be realized from this new ground state.

An example of a behavior type without a residual effect is `<point>`: usually, realization of a `<point>` behavior involves a final retraction phase that returns the ECA back to the ground state in which it was before starting realization of the `<point>` behavior.

A number of behavior types exist both in a version with and without residual effect. For example, after completion of a `<face>` behavior, the face of the ECA returns to the state it was in before the `<face>` behavior started, but a `<faceShift>` behavior will cause the face of the ECA to have a new ground state.

When both versions of a behavior are active at the same time, the version without residual effect has priority for being displayed, but the ground state is nevertheless changed by the behavior with residual effect.

Synchronization

For every behavior, its realization may be broken down into phases. Each phase is bounded by a sync-point that carries the name of the transition it represents, making it relatively straight-forward to align behaviors at meaningful boundaries (see Figure 4 for an example of the sync points for gestures). In the example below, the speech behavior and the point gesture are aligned at their start times.

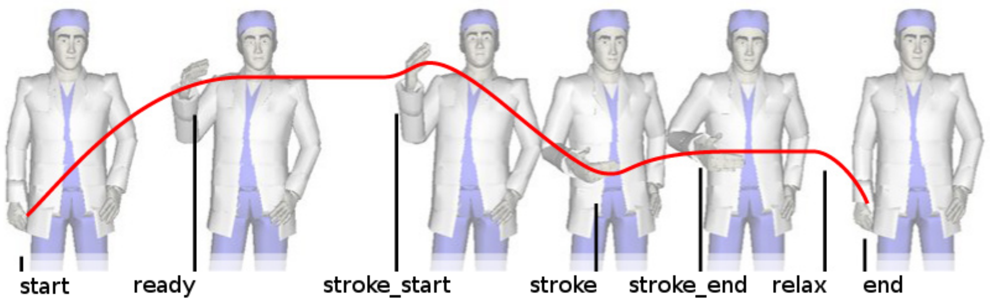


Figure 4: Synchronisation points for a gesture

Syntax

Synchronization is specified by assigning a syncref value to one or more of the sync attributes of a behavior. A syncref value is one of the following two forms:

[block_id:]behavior_id:sync_id [+/- offset]
A reference to a sync point of another behavior, optionally with a float offset in seconds. By default, this is a behavior in the same `<bml>` block that the syncref is contained in if optional prefix block_id is present, the syncref specifies a sync point of a behavior in the `<bml>` block with that ID.)

offset
A positive float offset in seconds relative to the start time of the surrounding `<bml>` block.

```
<!-- Timing example behaviors -->
<gaze start="0.3" end="2.14" /><!--absolute timing in seconds-->
<gaze stroke="behavior1:stroke" /><!--relative to another behavior -->
<gaze ready="behavior1:relax + 1.1" /><!--relative to another behavior, with offset-->
<gaze ready="bml3:behavior1:relax" /><!--relative to a behavior in another block-->
```

<constraint>

The `<constraint>` element provides a container for specifying additional constraints on the performance. BML 1.0 only defines three timing constraints:

- `<synchronize>` declares one or more sync points should be synchronized with a referenced sync-point notation
- `<before>` declares one or more sync points should be performed before a referenced sync-point notation
- `<after>` declares one or more sync points should be performed after a referenced sync-point notation

`<synchronize>`

`<synchronize>` constraints perform just like the sync-point attribute constraints, performing the sync-points of two or more behaviors at the same time.

```
<constraint>
   <synchronize>
      <sync ref="speech1:sync4"/>
      <sync ref="beat1:stroke:2"/>
      <sync ref="nod1:stroke"/>
   </synchronize>
</constraint>
```

This generalizes the attribute notation in three ways:

- A constraint can synchronize sync-points that do not have an attribute notation, such as speech word breaks and multi-stroke rhythmic gestures.
- A constraint can synchronize more than two behaviors to the same point.
- A constraint can remain optional (outside any `<required>` element) while the presence of the behaviors is still `<required>`.

<before>

<before> constrains one or more sync-points to perform before a specified sync-point notation.

```
<constraint>
   <before ref="speech_1:start">
      <sync ref="gaze_1:stroke"/>
   </before>
</constraint>
```

This constraint example requires the gaze_1 to acquire target (complete the stroke sync-point) before beginning speech_1.

`<after>`

`<after>` constrains one or more sync-points to perform before a specified sync-point notation.

```
<constraint>
   <after ref="speech_1:end+2">
      <sync ref="gaze_1:relax"/>
   </after>
</constraint>
```

This constraint example requires two seconds to pass after speech_1 completes before relaxing gaze_1. Extending `<constraint>`

We encourage BML developers to experiment with using the constraint element for the alternative functions through the use of namespaced elements and `<description>` extensions, for example:

- To specify some tolerance range for a synchronization operation.
- To specify a certain priority for a particular synchronization operation.
- To specify non-timing constraints such as modality

`<wait>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<wait>`
Sync Points	`start`, `end`
Attributes	`id`, `duration`, `start`, `end`
Contents	`none`

The `<wait>` element is a NO-OP behavior that facilitates flexible waiting times between behaviors.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <speech id="behavior1" start="0">
      <text>Good morning.</text>
   </speech>
   <wait id="behavior2" start="behavior1:end" duration="1"/>
   <speech id="behavior3" start="behavior2:end">
      <text>Goodbye.</text>
   </speech>
</bml>
```

Example: Wait for one second between the two speech fragments.

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular behavior. The id 'bml' is reserved.
duration	float	optional		the duration of the wait in seconds
start	float	optional		start of wait
end	float	optional		end of wait

Semantics

The synchronization constraints described above are all bidirectional. That is:

```
<head id="head1" stroke="gesture1:stroke" ... />
```

means that the strokes of head1 and gesture1 should be aligned. This synchronization constraint must be interpreted bidirectional: the exact same time constraint can be expressed by:

```
<gesture id="gesture1" stroke="head1:stroke" ... />
```

Default Sync Points and their Sync Attributes

All behaviors have sync points called start and end. Furthermore, for each behavior type a number of additional default sync points may be available. For every default sync point, the corresponding behavior XML element has a sync attribute of the same name.

New Sync Points

New sync points can be introduced for specific behavior types or description extensions. For example, in speech one can use the special `<sync>` tag to insert additional sync points in speech.

When new sync-points are introduced for a behavior, it is assumed that start and end will still refer to the first and last sync-point for that behavior.

Face Behaviours

The face can be controlled through various mechanisms. The `<faceLexeme>` behavior offers a range of predefined expressions such as "RAISE_EYEBROWS"; a limited set of mandatory lexemes is defined that should be offered by any BML Realizer. The optional `<faceFacs>` Core Extension allows precise control of the face in terms of the Facial Action Coding Scheme of Ekman. Finally, `<face>` and `<faceShift>` allow one to combine a set of partial expressions into one compound face expression, where the former is temporary, and the latter changes the BASE state of the ECAs face.

All face behavior types use the same set of sync points start, `attackPeak`, `relax`, and `end`. These sync points define a dynamic progress like in Figure 6 below. By using the Core Extension attribute `overshoot`, one can use these same alignment points to achieve a dynamic progress like in Figure 7, where `attackPeak` is the peak point of the initial overshoot of the face expression.

Figure 6: Onset/apex/offset dynamics specified using the sync points and the `amount` *attribute.*
Figure 7: Attack/sustain/release dynamics specified using the sync points and the `amount` *attribute plus the Core Extension* `overshoot` *attribute.*

Shared face attributes

Attributes

Attribute	Type	Use	Default	Description
id	ID	required		Unique ID that allows referencing to a particular behavior. The id 'bml' is reserved.
amount	float	optional	0.5	A float value between 0..1 to indicate the amount to which the expression should be shown on the face, 0 meaning 'not at all' and 1 meaning 'maximum, highly exaggerated'

Sync Attributes

Attribute	Description
start	Beginning of face expression
attackPeak	Maximum expression achieved
relax	Decay phase starts, not for `<faceShift>` behaviors!
end	Face expression ended, not for `<faceShift>` behaviors!

Overshoot Core Extension Attribute

Attribute	Type	Use	Default	Description
overshoot	float	optional	0	Fraction of overshoot of the attack peak, relative to amount (which defines the level of the sustain phase).

`<faceLexeme>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<faceLexeme>`
Sync Points	`start`, `attackPeak`, `relax`, `end`
Attributes	`id`, `lexeme`, `amount`, sync attributes (see above), `overshoot` (extension)
Contents	none

This behavior shows a (partial) face expression from a predefined lexicon. A faceLexeme is a convenience shorthand for combinations of more detailed low level face controls. The provided set of core lexemes allows one to perform face expressions using meaningful lexeme names, which are easier to learn than the (more detailed) Action Units provided by the `faceFacs` element.

Example: Raise both eye brows for 4 seconds.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0" character="Alice" id="bml1">
   <faceLexeme id="behavior1" lexeme="RAISE_BROWS" amount="0.8" start="0" end="4"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
lexeme	openSetItem	require		Member of a set of lexemes (See table below)

The following table shows suggested interpretations that a BML Realizer can use for the lexemes using Ekman's Facial Action Coding System. To offer the user more detailed control of the face, providing an implementation of the `<faceFacs>` element is suggested.

Lexeme	ACS equivalent
`OBLIQUE_BROWS`	AU1+AU4 both sides
`RAISE_BROWS`	AU1+AU2 both sides
`RAISE_LEFT_BROW`	AU1+AU2 left side
`RAISE_RIGHT_BROW`	AU1+AU2 right side
`LOWER_BROWS`	AU4 both sides
`LOWER_LEFT_BROW`	AU4 left side
`LOWER_RIGHT_BROW`	AU4 right side
`LOWER_MOUTH_CORNERS`	AU15 both sides
`LOWER_LEFT_MOUTH_CORNER`	AU15 left side
`LOWER_RIGHT_MOUTH_CORNER`	AU15 right side
`RAISE_MOUTH_CORNERS`	AU12 both sides
`RAISE_LEFT_MOUTH_CORNER`	AU12 left side
`RAISE_RIGHT_MOUTH_CORNER`	AU12 right side
`OPEN_MOUTH`	AU25+AU26
`OPEN_LIPS`	AU25
`WIDEN_EYES`	AU5+AU7
`CLOSE_EYES`	AU43

`<faceFacs>` (Core Extension)

Namespace	http://www.bml-initiative.org/bml/coreextensions-1.0
Element	`<faceFacs>`
Sync Points	start, attackPeak, relax, end
Attributes	id, au, side, amount, sync attributes (see above), overshoot (core extension)
Contents	none

This behavior provides control of the face through single Action Units from the Facial Action Coding Scheme. It is an Core Extension, that is, not every BML Compliant Realizer has to implement this behavior, but if a Realizer offers FACS based face control, they should adhere to the specification of this `<faceFacs>` behavior

A BML Compliant Realizer that implements this extension will provide at least the set of Action Units listed below. The other Action Units are not mandatory, but implementing the full set of Action Units is strongly recommended.

Example: Raise both eye brows for 4 seconds.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
   xmlns:ext="http://www.bml-initiative.org/bml/coreextensions-1.0"
       character="Alice"
       id="bml1">
   <ext:faceFacs id="behavior1" au="1" side="BOTH" amount="0.8" start="0" end="4"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
au	int	require	require	The number of the FACS Action Unit to be displayed
side	closedSetItem	optional		Which side of the face to display the action unit on. Possible values: [LEFT,RIGHT,BOTH] Note that for some Action Units, BOTH is the only possible value.

<face>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<face>`
Sync Points	`start, attackPeak, relax, end`
Attributes	`id`, `au`, `side`, `amount`, sync attributes (see above), `overshoot` (core extension)
Contents	`<lexeme>`, with attributes `lexeme` and `amount` that can take the same values as for the `<faceLexeme>` behavior. `<facs>`, with attributes `au`, `side` and `amount` that can take the same values as for the `<faceFacs>` element. (This child element is only available if FACS Core Extension is implemented; this `<facs>` child element has the same namespace as the `<faceFacs>` behavior element)

Compound behavior to specify the timing and alignment of several (partial) face expressions as one unit.

Example: Raise both eye brows for 4 seconds.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
    xmlns:ext="http://www.bml-initiative.org/bml/coreextensions-1.0"
       character="Alice"
       id="bml1">
   <face id="behavior1" amount="0.8" start="0" end="4">
      <ext:facs au="1" side="BOTH"/>
      <lexeme lexeme="WIDEN_EYES"/>
   </face>
</bml>
```

<faceShift>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<faceShift>`
Sync Points	`start`, `end`
Attributes	`id`, `amount`, sync attributes (see above)
Contents	`<lexeme>`, with attributes `lexeme` and `amount` that can take the same values as for the `<faceLexeme>` behavior. `<facs>`, with attributes `au`, `side` and `amount` that can take the same values as for the `<faceFacs>` element. (This child element is only available if FACS Core Extension is implemented; this `<facs>` child element has the same namespace as the `<faceFacs>` behavior element)

Compound behavior to specify the timing and alignment of several (partial) face expressions as one unit, where the specified compound face expression becomes the new BASE state of the ECAs face.

Example: Raise both eye brows for 4 seconds.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
    xmlns:ext="http://www.bml-initiative.org/bml/coreextensions-1.0"
       character="Alice"
       id="bml1">
   <faceShift id="behavior1" amount="0.8" start="0" end="4">
     <ext:facs au="1" side="BOTH"/>
     <lexeme lexeme="WIDEN_EYES"/>
   </faceShift>
</bml>
```

Gaze Behaviours

<gaze>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<gaze>`
Sync Points	`start`, `ready`, `relax`, `end`
Attributes	`id`, `target`, `influence`, `offsetAngle`, `offsetDirection`, sync attributes (see above)
Contents	none

This behavior causes the character to temporarily direct its gaze to the requested target. The influence parameter is read as follows: EYE means 'use only the eyes'; HEAD means 'use only head and eyes to change the gaze direction', etc.

Example: Direct the gaze towards the blue box for 9 seconds, using the eyes and the neck.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <gaze id="gaze1" start="1" end="10" influence="NECK" target="bluebox"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
target	targetID	require		A reference towards a target instance that represents the target direction of the gaze.
influence	openSetItem	optional		Determines what parts of the body to move to effect the gaze direction. [EYES, HEAD, SHOULDER, WAIST, WHOLE, ...]
offsetAngle	angle	optional	0.0	Adds an angle degrees offset to gaze direction relative to the target in the direction specified in the `offsetDirection`
offsetDirection	direction	optional	RIGHT	Direction of the offsetDirection angle [RIGHT, LEFT, UP, DOWN, UPRIGHT, UPLEFT, DOWNLEFT, DOWNRIGHT]

Sync Attributes

Attribute	Description
start	gaze starts to move to new target
ready	gaze target acquired
relax	gaze starts returning to default direction
end	gaze returned to default direction

`<gazeShift>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<gazeShift>`
Sync Points	`start`, `end`
Attributes	`id`, `target`, `influence`, `offsetAngle`, `offsetDirection`, sync attributes (see above)
Contents	none

This behavior causes the character to direct its gaze to the requested target. This changes the default state of the ECA: after completing this behavior, the new target is the default gaze direction of the ECA. The `influence` parameter is read as follows: EYE means 'use only the eyes'; HEAD means 'use only head and eyes to change the gaze direction', etcetera.

Example: Change the default gaze direction to be towards the blue box; the shift in gaze takes 1 second to be ready.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <gazeShift id="gaze1" start="1" end="2" influence="NECK" target="bluebox"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
target	targetID	require		A reference towards a target instance that represents the target direction of the gaze.
influence	openSetItem	optional		Determines what parts of the body to move to effect the gaze direction. [EYES, HEAD, SHOULDER, WAIST, WHOLE, ...]
offsetAngle	angle	optional	0.0	Adds an angle degrees offset to gaze direction relative to the target in the direction specified in the `offsetDirection`
offsetDirection	direction	optional	RIGHT	Direction of the offsetDirection angle [RIGHT, LEFT, UP, DOWN, UPRIGHT, UPLEFT, DOWNLEFT, DOWNRIGHT]

Sync Attributes

Attribute	Description
start	gaze starts to move to new target
end	gaze target acquired

Gesture Behaviours

`<gesture>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<gesture>`
Sync Points	`start`, `ready`, `strokeStart`, `stroke`, `strokeEnd`, `relax`, `end`
Attributes	`id`, `lexeme`, `mode`, sync attributes (see above)
Contents	none

Coordinated movement with arms and hands, recalled from a gesticon by requesting the corresponding lexeme.

Example: Make a waving gesture.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <gesture id="behavior1" lexeme="hello-waving" start="2"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
mode	closedSetItem	optional		What hand/arm is being used [LEFT_HAND, RIGHT_HAND, BOTH_HANDS]
lexeme	openSetItem	require		Refers to an animation or a controller to realize this particular gesture.Every realizer will offer at least this set of possible values: [BEAT]

The set of values for mode may in the future be extended with options such as HEAD or FOOT

Sync Attributes

Attribute	Description
start	beginning of gesture
ready	end of gesture preparation phase
strokeStart	start of the stroke
stroke	gesture stroke
strokeEnd	end of stroke
relax	start of retraction phase
end	end of gesture

`<pointing>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<pointing>`
Sync Points	`start`, `ready`, `strokeStart`, `stroke`, `strokeEnd`, `relax`, `end`
Attributes	`id`, `target`, `mode`, sync attributes (see above)
Contents	none

Deictic gesture towards the target specified by the target attribute.

Example: Point at the blue box.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <pointing id="behavior1" target="blueBox" mode="RIGHT_HAND" start="0" end="4"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
mode	closedSetItem	optional		What hand/arm is being used [LEFT_HAND, RIGHT_HAND, BOTH_HANDS]
target	targetID	require		The gesture is directed towards this target

Sync Attributes

Attribute	Description
start	beginning of gesture
ready	end of gesture preparation phase
strokeStart	start of the stroke
stroke	gesture stroke
strokeEnd	end of stroke
relax	start of retraction phase
end	end of gesture

Head Behaviours

<head>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<head>`
Sync Points	`start`, `ready`, `strokeStart`, `stroke`, `strokeEnd`, `relax`, `end`
Attributes	`id`, `lexeme`, `repetition`, `amount`, sync attributes (see above)
Contents	none

Movement of the head, recalled from a gesticon by requesting the corresponding lexeme. The stroke phase of the head motion (from `strokeStart` till `strokeEnd` is the ”meaningful” part of the head motion. The stroke sync point is the ”peak” moment of the motion. If `repetition > 1`, the meaning of the `stroke` sync point becomes undefined.

Example: Nod twice.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <head id="behavior1" lexeme="NOD" repetition="2" start="1" end="3"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id ’bml’ is reserved.
lexeme	openSetItem	require		Refers to an animation or a controller to realize this particular head behavior. Minimum set offered by all realizers: [NOD, SHAKE]
repetition	int	optional	1	Number of times the basic head motion is repeated.
amount	float	optional	1	How intense is the head nod? 0 means immeasurable small; 0.5 means ”moderate”; 1 means maximally large

The attribute speed has been discussed as possible extensions; however, they are not part of the Core 1.0 Standard.

Sync Attributes

Attribute	Description
start	start of the preparation phase
ready	end of the preparation phase
strokeStart	start of the stroke
stroke	stroke of the head motion. Note that the meaning of this sync point becomes undefined if `repetition > 1`
strokeEnd	end of stroke
relax	start of retraction phase
end	end of the head motion

`<headDirectionShift>`

Orient the head towards a target referenced by the target attribute.

Syntax

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<headDirectionShift>`
Sync Points	`start`, `end`
Attributes	`id`, `target`, sync attributes (see above)
Contents	none

The attribute speed has been discussed as possible extensions; however, they are not part of the Core 1.0 Standard.

Example: Orient the head towards TARGET1.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <headDirectionShift id="behavior1" target="TARGET1" start="2"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
target	targetID	require		Target towards which the head is oriented

Sync Attributes

Attribute	Description
start	Beginning of motion
end	Reached desired direction; set this direction as new BASE state

Locomotion Behaviour

`<locomotion>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<locomotion>`
Sync Points	`start`, `end`
Attributes	`id`, `target`, `manner`, sync attributes (see above)
Contents	none

This behavior causes the character to move to the requested target in the manner described (move the body of the character from one location to another.)

Example: Locomotion: walk to the audience.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
   <locomotion id="behavior1" target="AUDIENCE" manner="WALK"/>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
target	targetID	require		A reference towards a target instance that represents the end location of the locomotive behavior.
manner	openSetItem	optional		The general manner of locomotion [WALK, RUN, STRAFE ...] (WALK is the only mandatory element in the set)

Sync Attributes

Attribute	Description
start	Start of locomotion.
end	End of locomotion.

Posture Behaviours

BML allows one to specify *temporary* postures using `<posture>`, and permanent shifts to a new BASE posture using the `<postureShift>` behavior. Both behaviors have the same child elements to specify the form of the posture.

`<posture>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<posture>`
Sync Points	`start`, `ready`, `relax`, `end`
Attributes	`id`, sync attributes (see above)
Contents	`<stance>`, `<pose>`

Temporarily change the posture of the ECA. After the `<posture>` behavior has ended, return to the BASE posture.

Example: Crouch down with open arms for a short while.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0" character="Alice" id="bml1">
   <posture id="behavior1" start="5" end="15">
      <stance type="CROUCHING"/>
      <pose type="ARMS" lexeme="ARMS_OPEN"/>
   </posture>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.

Sync Attributes

Attribute	Description
start	Start moving to a new posture.
ready	New posture achieved
relax	start returning to BASE posture.
end	Temporary posture ended, back at BASE posture

<postureShift>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<postureShift>`
Sync Points	`start`, `end`
Attributes	`id`, sync attributes (see above)
Contents	`<stance>` (1), `<pose>` (unlimited)

Permanently change the BASE posture of the ECA.
Example: Sit down with arms crossed, and make that the new BASE posture.

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
      character="Alice"
      id="bml1">
  <postureShift id="behavior1" start="5">
    <stance type="SITTING"/>
    <pose type="ARMS" lexeme="ARMS_CROSSED"/>
  </postureShift>
</bml>
```

Attributes

Attribute	**Type**	**Use**	**Default**	**Description**
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.

Sync Attributes

Attribute	**Description**
start	Start moving to a new posture.
end	new BASE posture achieved.

`<stance>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<stance>`
Attributes	`type`
Contents	none

Child element of `<posture>` and `<postureShift>` behaviors, defines additions that modify the global body posture of the ECA. For each value of the part attribute, only one `<pose>` child is expected to be present. A BML Realizer may define any number of lexemes beyond the ones specified above.

Attributes

Attribute	Type	Use	Default	Description
type	closedSetItem	require		Global body posture. Possible values are [SITTING, CROUCHING, STANDING, LYING].

`<pose>`

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<pose>`
Attributes	`part`, `lexeme`
Contents	none

Child element of `<posture>` and `<postureShift>` behaviors, defines additions that modify the global body posture of the ECA. For each value of the part attribute, only one `<pose>` child is expected to be present. A BML Realizer may define any number of lexemes beyond the ones specified above.

Attributes

Attribute	**Type**	**Use**	**Default**	**Description**
part	openItemSet	require		What part of the body is affected? Possible values are [ARMS, LEFT_ARM, RIGHT_ARM, LEGS, LEFT_LEG, RIGHT_LEG, HEAD, WHOLEBODY].
lexeme	openItemSet	require		What configuration is set to the given part? Some possible values are [ARMS_AKIMBO, ARMS_CROSSED, ARMS_NEUTRAL, ARMS_OPEN, LEGS_CROSSED, LEGS_NEUTRAL, LEGS_OPEN, LEANING_FORWARD, LEANING_BACKWARD, ...]

Speech Behaviours

<speech>

Namespace	http://www.bml-initiative.org/bml/bml-1.0
Element	`<speech>`
Sync Points	`start`, `end`, any `sync` element in the speech (see below)
Attributes	`id`, sync attributes (see above)
Contents	exactly one `<text>` child containing the text to be spoken, which in turn may contain one or more `<sync>` markers. A `<sync>` marker has an attribute id of type ID, the value of which is unique within the context of this `<speech>` element.

Utterance to be spoken by a character. Realization of the ¡speech¿ element generates both speech audio (or text) and speech movement, for example using a speech synthesizer and viseme morphing. The `<speech>` element requires a sub-element. This sub-element is a `<text>` element that contains the text to be spoken, with optionally embedded `<sync>` elements for alignment with other behaviors.

Example: This is an example of a complete speech behavior, synchronized to a beat gesture (striking on "speech").

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0" character="Alice" id="bml1">
   <speech id="speech1" start="6">
      <text>This is a complete
      <sync id="syncstart1" /> BML core speech description.
      </text>
   </speech>
</bml>
```

Attributes

Attribute	Type	Use	Default	Description
id	ID	require		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.

Sync Attributes

Attribute	Description
start	Start of speech.
end	End of speech.

SSML Core Description Extension

The SSML description extension is of type ”application/ssml+xml”.
Its namespace is ”http://www.w3.org/2001/10/synthesis”.

The format of the content of this extension is defined at http://www.w3.org/TR/speech-synthesis/

Example: Using the SSML Description extension for speech

```
<speech id="s1">
   <text>Hello! This is a basic SSML <sync id="bml"/>BML test.</text>
   <description priority="2" type="application/ssml+xml">
      <speak xmlns="http://www.w3.org/2001/10/synthesis">
      Hello! <break time="3s"/> <prosody pitch="high">This is a basic SSML <mark name="bml"/>BML
          test</prosody>.
      </speak>
   </description>
</speech>
```

MaryXML Core Description Extension

The MaryXML description extension is of type ”maryxml”.
Its namespace is ”http://mary.dfki.de/2002/MaryXML”.
It allows one to specify more detail for the TTS engine, if one uses [[http://mary.dfki.de MaryTTS]] for speech generation.

The format of the content of this extension is defined at http://mary.dfki.de/documentation/maryxml

Example: Using the SSML Description extension for speech

```
 <speech id="s1">
   <text>Hello! This is a basic Mary <sync id="bml"/>BML test.</text>
   <description priority="2" type="maryxml">
      <maryxml xmlns="http://mary.dfki.de/2002/MaryXML">
      Hello! This is a basic Mary <mark name="bml"/>BML test.
      </maryxml>
   </description>
</speech>
```

Description Extensions and Other Extension Mechanisms

The core BML behavior elements are by no means comprehensive, and much of the ongoing work behind BML involves identifying and defining a broad and flexible library of behavior (types). Implementors are encouraged to explore new behavior elements and more detailed ways to specify existing core behaviors. BML allows such extensions in several ways:

- Additional behaviors should be designed as new XML elements using custom XML namespaces.
- Specialized attributes can be used to extend core BML behaviors. Such attributes should be identified as non-standard BML by utilizing XML namespaces.
- Behavior Description Extensions provide a principled way of specifying core BML behaviors in a more detailed manner, typically using existing XML languages for that specific behavior.

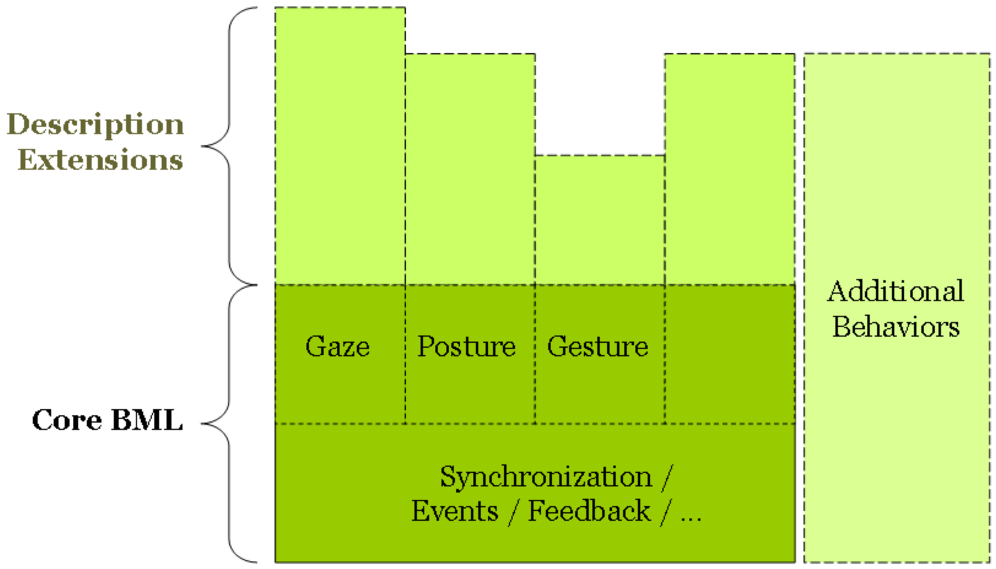


Figure 5: Extending BML

The following example utilizes a customized `animation` behavior and a customized `joint-speeds` attribute. The latter specifies the core gaze behavior in a more detailed manner. Both extensions are from the SmartBody project.

Example: Using extensions

```
<bml xmlns="http://www.bml-initiative.org/bml/bml-1.0"
    xmlns:sbm="http://www.smartbody-anim.org/sbm">
    <gaze id="gaze1" target="AUDIENCE" sbm:joint-speeds="100 100 100 300 600"/>
    <sbm:animation name="CrossedArms_RArm_beat"/>
  </bml>
```

If a realizer cannot interpret extended BML, it should deal with it in the way suggested in the Section Failure and Fallback.

Behavior Description Extensions

BML allows for additional behavior descriptions that go beyond the core BML behavior specification in describing the form of a behavior. Additional descriptions are embedded within a behavior element as children elements of the type description. The type attribute of the description element should identify the type of content, indicating how it should be interpreted. Even if additional descriptions are included in a behavior, the core attributes of the behavior element itself cannot be omitted since the core specification is always the default fall-back.

Description elements in BML can include existing representation languages such as SSML, Tobi, etc. or new languages can be created that make use of advanced realization capabilities. Each description element should be a self-contained description of a behavior because a behavior realizer may not know how to combine multiple behavior descriptions. It is required that each description provides exactly the same synchronization points as its accompanying core BML. It is however allowed to place the synchronization points in the description extension at slightly different positions than those in the core BML. This can be used to, for example, to provide synchronization at syllable level rather than a word level in a description extension of a speech behavior.

If a realizer does not known how to interpret the available description types, it should default to the core behavior.

If multiple description elements are given, and a realizer is capable of interpreting more than one, the realizer should use the highest priority description.

Example: use an audio file to play back this speech behavior. If that's not supported, use SSML. As a last resort, fall back to the core behavior. Note that the descriptions specify the same sync points as the core behavior.

Examples

```
<speech id="s1">
   <text>This is the proposed BML <sync id="tm1"/> extended speech specification.</text>
   <description priority="1" type="application/ssml+xml">
      <ssml:speak xmlns:ssml="http://www.w3.org/2001/10/synthesis">
      This is the <ssml:emphasis>proposed</ssml:emphasis> BML <ssml:mark name="tm1"/> extended
          speech specification.
      </ssml:speak>
   </description>
   <description priority="3" type="audio/x-wav">
      <audio:sound xmlns:audio="http://www.ouraudiodesc.com/">
         <audio:file ref="bml.wav"/>
         <audio:sync id="tm1" time="2.3" />
      </audio:sound>
   </description>
</speech>
```

Example: Using description extensions for speech

```
<speech id="s1">
   <text>This is the proposed BML <sync id="tm1"/> extended speech specification.</text>
   <description priority="1" type="application/ssml+xml">
      <speak xmlns="http://www.w3.org/2001/10/synthesis">
      This is the <emphasis>proposed</emphasis> BML <mark name="tm1"/> extended speech
          specification.
      </speak>
   </description>
   <description priority="3" type="audio/x-wav">
      <sound xmlns="http://www.ouraudiodesc.com/">
         <file ref="bml.wav"/>
         <sync id="tm1" time="2.3" />
      </sound>
   </description>
</speech>
```

Example: A slightly less verbose example of the same behavior, using default namespaces for audio and SSML.

Failure and Fallback

When a realizer is unable to interpret or execute part of a `<bml>` block, it should deal with it in the following ways.

- if unable to execute `<required>` block: drop complete `<bml>` block; send warning feedback
- if unable to execute a behavior child: drop behavior, send warning feedback
- if unable to adhere to a constraint specified in an attribute in a behavior child: drop behavior, send warning feedback
- if unable to interpret a description extension: fallback to lower priority description extension, or to core behavior
- if unable to interpret extended behaviors: drop behavior, send warning feedback
- if unable to interpret extended attributes: drop attribute, send warning feedback

Feedback

A BML realizer should provide a behavior planner with various types of feedback. Progress feedback gives information on the execution status of ongoing behaviors. Prediction feedback provides the ”scheduling solution” of behaviors, such as the expected timing of sync points. Warning feedback indicates that the execution or scheduling of some behavior(s) failed, or that some time constraints could not be achieved.

Prediction Feedback

Name	Prediction Feedback
Status	Optional
XML Namespace	http://www.bml-initiative.org/bml/bml-1.0

Prediction feedback provides information about the expected realization of the `<bml>` request. It consists of block prediction, and behavior prediction feedback. Block prediction feedback contains information on the global start and end time of a block. Behavior prediction feedback contains information on the local timing of all sync points of the behavior.

Prediction feedback may be revised – later feedback counts as a ’revision’ overriding all previous prediction feedback concerning the same `<bml>` block or the same behavior element.

Syntax

The syntax is similar to that of the BML blocks. The prediction feedback is wrapped into a `<predictionFeedback>` element, which has an optional `characterId` attribute indicating the ID of the character.

Example: Block prediction example

```
<predictionFeedback characterId="doctor" (optional attribute)>
  <bml id="bml1" globalStart="1" globalEnd="30"/>
</predictionFeedback>
```

Example: Behavior prediction example

```
<predictionFeedback>
  <gesture id="bml1:gesture1" lexeme="BEAT" start="0" ready="1" strokeStart="3" stroke="4"
      strokeEnd="5" relax="6" end="7"/>
</predictionFeedback>
```

Example: Solution for speech, internal syncs are resolved by a time attribute in the sync tag.

```
<predictionFeedback>
 <speech start="0" ready="0" strokeStart="0" stroke="4" strokeEnd="4" relax="4" end="4"
     id="bml1:speech1">
  <text>Hello <sync id="s1" time="2"/> world</text>
 </speech>
</predictionFeedback>
```

Example: Solution for a behavior with a custom sync point. The prediction for the timing of the custom sync point is provided in an embedded the sync tag.

```
<predictionFeedback>
  <gesture id="bml1:gesture1" type="LEXICALIZED" lexeme="CUSTOM_LEXEME" start="0" ready="0"
      strokeStart="0" stroke="4" strokeEnd="4" relax="4" end="4">
    <sync id="customsync1" time="3"/>
  </gesture>
</predictionFeedback>
```

Example: A prediction feedback may contain multiple behaviors or blocks.

```
<predictionFeedback>
  <bml id="bml1" globalStart="1" globalEnd="7"/>
  <gesture id="bml1:gesture1" lexeme="BEAT" start="0" ready="1" strokeStart="3" stroke="4"
      strokeEnd="5" relax="6" end="7"/>
  <head id="bml1:head1" lexeme="NOD" start="0" ready="1" strokeStart="3" stroke="4" strokeEnd="5"
      relax="6" end="7"/>
</predictionFeedback>
```

Shape Feedback

The behaviors elements within a prediction feedback may be used to provide the behavior planner with information on the shape of a to be executed behavior. For example, the BML block:

```
<bml id="bml1">
   <gesture id="b1" lexeme="BEAT"/>
</bml>
```

may result in feedback of the form:

```
<predictionFeedback>
   <gesture id="b1" lexeme="BEAT" mode="RIGHT_HAND"
      start="0" ready="1"
      strokeStart="1" strokeEnd="2"
      relax="2" end="3"/>
</predictionFeedback>
```

In addition to informing the behavior planner on the timing of gesture b1, this feedback message also informs the behavior planner that the realizer chose to execute the beat gesture with the right hand. When desired, description extensions can be employed to provide very detailed shape information.

Multiple Revisions

Prediction feedback may be revised – later feedback counts as a 'revision' overriding all previous prediction feedback concerning the same `<bml>` block or the same behavior element. As such, the feedback can be used as (potentially continually updated) predictions of the timing of behaviors.

Maximum Information

The BML Realizer must send information about all sync points that it does know about. If it does not know, it will leave out the sync point from the returned BML expression.

Progress Feedback

Provides real-time information on the progress of ongoing behavior. Consists of progress feedback on the `<bml>` block and individual sync point level.

Name	Progress Feedback
Status	Mandatory
XML Namespace	http://www.bml-initiative.org/bml/bml-1.0

`<blockProgress>`

Block start block start contains the following attributes:

Attribute	Type	Use	Default	Description
id	ID	required		Unique ID that allows referencing to a particular `<bml>` behavior. The id 'bml' is reserved.
globalTime	float	required		Global time stamp
characterId	ID	optional		ID of the character to which the feedback belongs

Example: block start feedback

```
<blockProgress id="bml1:start" globalTime="10" characterId="doctor"/>
```

Example: block end feedback

```
<blockProgress id="bml1:end" globalTime="15" characterId="doctor"/>
```

BML compliant realizers should provide progress feedback for at least the start and end of the BML block (the format for this is shown in the examples above). Optionally, realizers might provide feedback on other time events of a BML block. For example: a realizer might indicate that it is subsiding (all behavior in the block is either ended or in a relax phase) in the following manner:

```
<blockProgress id="bml1:relax" globalTime="14" characterId="doctor"/>
```

`<syncPointProgress>`

Sync point progress feedback contains the following attributes:

Attribute	**Type**	**Use**	**Default**	**Description**
id	ID	required		full ID of the sync point to which the feedback belongs
globalTime	float	required		Global time stamp of when the sync point happened
time	float	required		Local time stamp of when the sync point happened, relative to the block start of the corresponding ¡bml¿ block
characterId	ID	optional		ID of the character to which the feedback belongs

Example: sync point progress feedback

```
<blockProgress id="bml1:end" globalTime="15" characterId="doctor"/>
```

The Order of Progress Feedback

Some order constraints are enforced upon the sending of progress feedback:

- The block start feedback of a `<bml>` block should occur before all sync point progress feedback messages of all behaviors in the block.
- The block end feedback of a `<bml>` block should occur after all sync point progress feedback messages of all behaviors in the block.
- The sync point progress feedback of behaviors should arrive in the default order. For example, if a `start` and `ready` sync of a behavior occur at the same time, the sync point progress feedback of the start sync should still be sent before that of the `ready` sync.

Warning Feedback

Warning feedback notifies the behavior planner that requested behaviors and/or constraints have failed to realize, and possibly led to aborting the performance.

It contains the following information:

- BML ID
- warning type [BML parsing failure, no such gaze/walk/point target, impossible to schedule, realizer does not support behavior type x, realizer cannot construct behavior type x, ...]
- Whether the block was interrupted as a whole or the id of the behavior that failed
- *Optional*: a human-readable description of the error
- *Optional*: character id

```
<warningFeedback id="bml1" characterId="armandia" type="PARSING_FAILURE">
Cannot parse BML block
</warningFeedback>
```

bml1 fails as a whole

```
<warningFeedback id="bml1:gaze1" characterId="armandia" type="NO_SUCH_TARGET">
"doctor" is not a valid gaze target.
</warningFeedback>
```

behavior gaze1 in bml1 fails

The content of the `<warningFeedback>` element is left open. In the examples we show how human readable error messages could be embedded in warning feedback. Alternatively, realizers could embed a custom XML element that describes the warning in more detail.

The following feedback types are included in BML 1.0:

`PARSING_FAILURE`	There is an error in the syntax of the `<bml>` block
`NO_SUCH_TARGET`	locomotion/gaze/.. target does not exist in the world
`IMPOSSIBLE_TO_SCHEDULE`	The BML block contains conflicting constraints(e.g. `beh1:start=beh1:end+1`)
`BEHAVIOR_TYPE_NOT_SUPPORTED`	The realizer does not support a core behavior type requested in the BML block (e.g. when a realizer steer a head only avatar is asked to do a locomotion behavior)
`CUSTOM_BEHAVIOR_NOT_SUPPORTED`	The realizer does not support a custom (non core) behavior
`CUSTOM_ATTRIBUTE_NOT_SUPPORTED`	The realizer does not support a custom attribute specified on a core behavior
`CANNOT_CREATE_BEHAVIOR`	The realizer cannot construct a behavior (given specified time constraints and shape attributes)

Contributors

Over the years, a large number of people have contributed to the papers, workshops and developer meetings leading to this standard. Below, you find an (incomplete) list of names.

Aleksandra Cerekovic, Alex Hill, Alexis Heloir, Andrew Marshall, Ari Shapiro, Brigitte Krenn, Catherine Pelachaud, Dan Loehr, Dennis Reidsma, Hannes Högni Vilhjálmsson, Hannes Pirker, Herwin van Welbergen, James Gruber, Job Zwiers, John Borland, Jon Homer, Justine Cassell, Kristinn R. Thórisson, Marco Vala, Maurizio Mancini, Michael Kipp, Michael Krieger, Michael Neff, Michael Wißner, N. Cantelmo, N.E. Chafai, Nicolas Schulz, Paul Tepper, Prasan Samtani, Quoc Anh Le, Radek Niewiadomski, Rick van der Werf, Stacy Marsella, Stefan Kopp, Tim Bickmore, W. Lewis Johnson, Zsofia Ruttkay