BEHAVIOR GENERATION FOR INTERPERSONAL COORDINATION WITH VIRTUAL HUMANS

ON SPECIFYING, SCHEDULING AND REALIZING MULTIMODAL VIRTUAL HUMAN BEHAVIOR

HERWIN VAN WELBERGEN

PhD dissertation committee:

Chairman and Secretary:
Prof. dr. ir. A. J. Mouthaan, Universiteit Twente, NL
Promotor:
Prof. dr. ir. A. Nijholt, Universiteit Twente, NL
Assistant-promotors:
Dr. ir. D. Reidsma, Universiteit Twente, NL
Dr. Zs. M. Ruttkay, Moholy-Nagy Művézeti Egyetem, HU
Members:
Prof. dr.-ing. S. Kopp, Universität Bielefeld, DE
Dr. M. Neff, UC Davis, CA, US
Prof. dr. ir. P. H. Veltink, Universiteit Twente, NL
Prof. dr. R. C. Veltkamp, Universiteit Utrecht, NL
Dr. J. Zwiers, Universiteit Twente, NL
Paranymphs:
M. Knol
B. van Straalen, MSc.

CTIT Dissertation Series No. 11-202
Center for Telematics and Information Technology (CTIT)
P.O. Box 217 – 7500AE Enschede – the Netherlands

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

Human Media Interaction
The research reported in this thesis has been carried out at the Human Media Interaction research group of the University of Twente.

SIKS Dissertation Series No. 2011-24
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 978-90-365-3233-4
ISSN: 1381-3617, No. 11-202

BEHAVIOR GENERATION FOR INTERPERSONAL COORDINATION WITH VIRTUAL HUMANS

ON SPECIFYING, SCHEDULING AND REALIZING MULTIMODAL VIRTUAL HUMAN BEHAVIOR

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee
to be publicly defended
on Friday, September 9, 2011 at 16.45

by

Herwin van Welbergen

born on October 23, 1980

in Deventer, The Netherlands

This thesis has been approved by:

Promotor:

Prof. dr. ir. A. Nijholt

Assistant-promotors:

Dr. Zs. M. Ruttkay
Dr. ir. D. Reidsma

ISBN: 978-90-365-3233-4

Acknowledgments

> On thee thou must take a long journey
> Therefore thy book of count with thee thou bring;
> For turn again thou can not by no way,
> And look thou be sure of thy reckoning
>
> Everyman (early 16th century translation of the Dutch morality play 'Elckerlyc')

In this thesis I state that building a virtual human requires expertise in many research areas and is probably not an effort that can be undertaken by a single research group. I was lucky enough to work together with several experts in different fields. This makes this thesis truly reflect the collaboration required to build (parts of) a virtual human. Chapter 2 reflects Ben van Basten's and my efforts to make sense of and organize the computer animation literature and reflects the some of the field knowledge of Arjan Egges, Zsófi Ruttkay and Mark Overmars. The first part of Chapter 6 describes the Behavior Markup Language, a joint effort of the SAIBA initiative, consisting of over 20 researchers (including myself) of over seven different research institutes. Several people have contributed to the development of Elckerlyc (Chapter 8). Dennis Reidsma contributed to the design, implementation and documentation of many of Elckerlyc's features. Job Zwiers contributed to Elckerlyc's design and implemented many of the libraries used by Elckerlyc (e.g. for quaternion/vector/matrix math, animation, rendering, XML parsing). Ronald Paul devised the first version of Elckerlyc's BML parser and created the basis of what is now Elckerlyc's facial animation system. Hendri Hondorp manages many of the tools that are used to design, document and test Elckerlyc, including the svn server, the continuous integration system and Elckerlyc's current website. Daniel Davison is the newest addition to the Elckerlyc team. He created the PictureEngine of Chapter 10.1.3.1 and will contribute to Elckerlyc's further modularization. Elckerlyc's design was partly motivated by the needs of its users, including Mark ter Maat, Ronald Poppe, Guillermo Solano, Eike Dehling, Rieks op den Akker, Randy Klaassen en Jan van der Meij. Chapter 10.3 highlights some of the experiments conducted and applications developed by these users. The eNTERFACE project "Continuous interaction for ECAs" in which a multi-disciplinairy team (Khiet Truong, Iwan de Kok, Daniel Neiberg, Sathish Pammi, Dennis Reidsma, Bart van Straalen and myself) attempted to build an attentive speaker was especially influential in shaping several of Elckerlyc's design features. Chapter 9 presents joint work with the SmartBody team

(Ari Shapiro, Yuyu Xu, Marcus Thiebaux, Wei-Wen Feng, Jingqiao Fu) at ICT/USC on testing BML Realizers. Some of the work I did during my PhD did not fit the this thesis. I worked together with Sander Jansen to define and try out a methodology to measure the naturalness of computer animation, based on an analysis technique suggested to me by Rob van de Lubbe. Thanks to you both, I hope to find some time in the future to actually make use of this methodology. I also thoroughly analyzed some recordings of Wim clapping his hands. I don't think that that analysis will ever serve any good purpose. It might be amusing to retarget the recordings to a virtual walrus.

I would like to thank my committee for their participation in my defense, especially Michael Neff who went out of his way to provide very detailed comments and posed many interesting questions. I hope I was able to address and answer them all. The ESF team provided the necessary subtraction from work and several nice places to visit during holidays (thanks Damien!). Andrew made me this amazing cover, thanks again for that.

Tot zover alle credits en Engelse bedankjes, nu over naar het Nederlands. Als eerste wil ik graag mijn grote groep begeleiders bedanken. Zsófi begeleide me in de eerste twee jaar van mijn aio-schap. Haar aanstekelijke enthousiasme zorgde voor vele interesante ideeën (en artikelen over deze ideeën). Ze zorgde ervoor dat ik in mijn eerste jaar alle relevante conferenties had bezocht en bijna alle mensen in het veld had ontmoet. Job zorgde voor de scherpe kritiek die menig ontwerp en paper flink heeft verbeterd. Daarnaast was hij beschikbaar om me te helpen met ingewikkelde conceptuele of wiskundige problemen. Dennis nam mijn begeleiding van Zsófi's over in de tweede helft van mijn aioschap. Bij Dennis kon ik altijd even binnenvallen om een stukje ontwerp of een half idee te bespreken. Na zo'n gesprek kwam ik altijd wijzer terug. Zijn begeleiding kwam vaak in de vorm van een intensieve samenwerking. Veel van de ideeën, ontwerpen en implementaties in het tweede deel van mijn proefschrift komen voort uit deze samenwerking. Daarnaast heb ik veel van Dennis geleerd over schrijven en lesgeven. Anton gaf me het vertrouwen om een jaar en negen maanden voordat het geld echt binnen was al aan de slag te kunnen, volgens mij heb ik hem daar nooit voor bedankt. Bij deze alsnog.

De HMI groep was een inspirerende en prettige groep in om in te werken. Ik hoop dan ook de samenwerking met HMI te kunnen voortzetten in de toekomst. Lynn, bedankt voor het zeer nauwkeurig verbeteren van mijn Engels, ik heb er hopelijk wat van geleerd. Charlotte en Alice, bedankt voor jullie hulp in allerlei administratieve zaken. Hendri, bedankt voor alle technische ondersteuning, je hebt me erg geholpen om mijn werk soepel te kunnen doen. In het bijzonder wil ik mijn (oud)kamergenoten Wim, Thijs, Ivo en Bart bedanken. Ik heb met jullie zowel op het werk als daarbuiten veel lol gehad. In navolging van Ivo zal ik het ook niet over een aantal dingen hebben, bijvoorbeeld over Thijs' slechte (maar wel erg vermakelijke) kantoorhumor, Wim's neiging om alle HMI apparatuur op zijn bureau te verzamelen en open te schroeven, Bart die het vermijden van ochtenden tot een ware kunst heeft verheven en Ivo, wiens onderzoek kleine meisjes aan het huilen maakt. Heren, het ga jullie allemaal erg goed, en ik hoop dat we ook na mijn promotie nog contact zullen houden.

Aan het begin van mijn aio-schap was ik een veel geziene gast bij Mark Overmars' Games and Virtual Worlds groep. Het was altijd er plezierig om daar wat te brainstormen over computer animatie ideeën met Arjan en vooral Ben en om, als ik dan toch in Utrecht was, weer eens een biertje te pakken met Michiel. Met de schakers van Drienerlo en Drienerlo in de Nahand heb ik me de afgelopen jaren vreselijk vermaakt, zowel achter als naast het bord. Bedankt voor de belachelijke partijen, wijze schaak dogmas en vele liters bier. Martijn, bedankt voor meer dan twintig jaar vriendschap, het is een eer om je als paranymph bij mijn promotie te hebben. Tot slot, pa en ma, bedankt voor de ondersteuning door de jaren heen en jullie aandringen om toch maar snel een baan te zoeken. Dat is gelukt.

Herwin van Welbergen
Enschede, August 2011

Summary

Virtual environments inhabited by virtual humans are now commonplace in many applications, particularly in (serious) games. These virtual humans interact with other (virtual) humans and their surroundings. For such interactions, detailed *control* over their behavior is crucial. The control requirements for virtual humans range from providing physical interaction with the environment to providing tight coordination with a human interaction partner. Furthermore, the behavior of virtual humans should *look* realistic. Throughout this thesis the term *naturalness* is used for such perceived realism.

Many techniques achieve real-time animation. These techniques differ in the trade-off they offer between the control that can be exerted over the motion, the motion naturalness, and the required calculation time. Choosing the right technique depends on the requirements of the application it is used in. Motion (capture) editing techniques employ the detail of captured motion or the talent of skilled animators, but they allow little deviation from the captured examples and can lack physical realism. Procedural motion offers detailed and precise control using a large number of parameters, but lacks naturalness. Physical simulation provides integration with the physical environment and physical realism. However, physical realism alone is not enough for naturalness and physical simulation offers poor precision in both movement timing and limb placement. Hybrid animation techniques combine and concatenate motion generated by different animation paradigms to enhance both naturalness and control.

This thesis contributes one such hybrid technique: mixed dynamics. It combines the physical naturalness provided by physically realistic animation with the control provided by procedural animation. It builds on the notion that the requirements of physical integrity and tight temporal synchronization are often of different importance for different body parts. For example, for a gesturing virtual human, tight synchronization with speech is primarily important for arm and head movement. At the same time, a physically valid balancing motion of the whole body could be achieved by moving only the lower body, where precise timing is less important. Mixed dynamics allows one to mix procedural arm and head gestures with physical simulation of the rest of the body. The forces generated by the gesturing body parts are transferred to the physically simulated body parts, thus creating whole body animation that appears to respect the laws of physics in a believable manner and that is internally coherent (that is: the movement of the physically steered body parts is affected by the movement of the procedurally steered ones).

Traditionally, interaction with virtual humans was designed using ‘transmitter/

receiver' interaction paradigms, in which the user and the virtual human take turns to transmit (encode) and receive (decode) messages carrying meaning that travel across channels between them. Such an interaction model is insufficient to capture the richness of human-human interaction (including conversation). Natural interaction requires a *continuous* interaction paradigm, where actors perceive acts and speech of others continuously, and where actors can act continuously, simultaneously and therefore overlapping in time. Such continuous interaction requires that the perception capabilities of the virtual human are fast and provide incremental interpretation of another agent's behavior. These interpretations are possibly extended and revised over time. To be able to deal with such continuously updated interpretations and rapid observations, the multimodal output generation modules of the virtual humans should be capable of flexible production of behavior. This includes adding or removing behavior elements at a late time, coordinating behavior with predicted interlocutor events and adapting behavior elements that have already been scheduled or are currently playing. This thesis deals with the specification and execution of such flexible multimodal output.

The Behavior Markup Language (BML) has become the de facto standard for the specification of the synchronized motor behavior (including speech and gesture) of virtual humans. BML is interpreted by a *BML Realizer*, that executes the specified behavior through the virtual human it controls. Continuous interaction applications with virtual humans pose several generic requirements on the specification of behavior execution, beyond that of multimodal internal (that is, within the virtual human) synchronization and form descriptions provided by BML. Continuous interaction requires specification mechanisms for the interruption of ongoing behavior, the change of the shape of ongoing behavior (e.g. speak louder) and the synchronization of behavior with predicted external time events (e.g. originating from the interlocutor). This thesis contributes BML Twente (BML$^{\mathrm{T}}$), a language that extends BML by providing the *specification* of the continuous interaction capabilities discussed above. It thus provides a generic interface to a Realizer through which continuous interaction can be realized.

"Elckerlyc" is designed as a BML Realizer for generating multimodal verbal and nonverbal behavior for virtual humans.[1] The main design characteristics of Elckerlyc are that (1) it is designed specifically for *continuous interaction* with tight coordination between the behavior of a virtual human and that of its interaction partner; (2) it provides an *adjustable trade-off between the control and naturalness* offered by different animation paradigms (e.g. procedural body animation and physical body animation; MPEG-4 facial animation and morph-based facial animation), allowing the execution of the paradigms simultaneously; and (3) it is designed to be highly *modular and extensible* and allows adaptations and extensions of the capabilities of the virtual human, without having to make invasive modifications to Elckerlyc itself.

A BML Realizer is responsible for executing the behaviors specified in the BML blocks sent to it, in such a way that the time constraints specified in the BML blocks

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

are satisfied. Realizer implementations, including Elckerlyc, handle this by separating the BML scheduling process from the behavior execution process. The scheduling process is responsible for creating a multimodal behavior plan that is in a suitable form for execution.

In most BML Realizers the scheduling of BML results in a *rigid* multimodal realization plan in which the timing of all behaviors is fixed. In Elckerlyc however, continuous interaction requirements dictate a multimodal behavior plan that is modified continually at execution time. Such modifications should not invalidate the time constraints between, for example, speech and gesture that are specified in BML or result in biologically infeasible behavior. Elckerlyc contributes a *flexible* multimodal plan representation that allows plan modification, while retaining timing and naturalness constraints.

Elckerlyc is the first BML Realizer specifically designed for continuous interaction. It contributes flexible formalisms for both the specification and the modification of running behavior. It pioneers the use of physical simulation and mixed dynamics in a real-time multimodal virtual human platform. This provides physically coherent whole body involvement, a naturalness feature that is lacking in virtual human platforms that solely use procedural animation. Furthermore, Elckerlyc provides a more extensible and more thoroughly tested architecture than existing BML Realizers. Other Realizers have implemented alternative and more elaborate scheduling algorithms, or provide motor control on modalities that are not present in Elckerlyc (e.g. blushing), or provide specialized behavior elements (e.g. walking). Elckerlyc's extensibility allows one to easily implement such specialized behaviors on existing modalities or new modalities into Elckerlyc. Elckerlyc was also designed to allow the use of new scheduling algorithms; the feasibility of this design feature is yet to be proven.

Elckerlyc is employed in several virtual human applications. Several of its design features were motivated, fine-tuned and finally demonstrated by this 'field' experience of Elckerlyc.

Samenvatting

Virtuele omgevingen, bevolkt door virtuele mensen, worden gebruikt in verscheidene applicaties, waaronder (serious) games. Deze virtuele mensen interacteren met andere (virtuele) mensen en met hun omgeving. Voor deze interacties is het van cruciaal belang dat virtuele mensen op gedetailleerd niveau controle te kunnen uitoefenen op hun gedrag. Het gedrag van deze virtuele mensen moet kunnen worden geregeld op verschillende niveaus, van fysische interactie met de omgeving tot strakke cordinatie met het gedrag van een (menselijke) gesprekspartner. Bovendien moet het gedrag van virtuele mensen er realistisch uitzien. In deze samenvatting gebruik ik de term *natuurlijkheid* voor zulk waargenomen realisme.

Een groot aantal technieken kan gebruikt worden voor real-time animatie. Deze technieken bieden verschillende trade-offs tussen de controle die kan worden uitgeoefend over de beweging, de natuurlijkheid van de beweging en de benodigde rekentijd. Een passende animatie techniek wordt gekozen aan de hand van de vereisten van de applicatie waarin hij nodig is. Motion capture bewerkings technieken gebruiken het detail van opgenomen beweging, of het talent van animatie artiesten. Motion capture bewerkingstechnieken laten slechts weinig afwijking van de opgenomen beweging toe en fysisch realisme wordt niet altijd bereikt. Procedurele animatie biedt gedetailleerde en precieze controle over beweging, waarbij een groot aantal parameters gebruikt kan worden om deze te specificeren. Deze controle gaat ten koste van de natuurlijkheid van de animatie. Fysische simulatie biedt integratie met de fysische omgeving en fysisch realisme. Echter, fysisch realisme alleen is niet afdoende voor natuurlijkheid en fysische simulatie biedt slechte precisie in zowel bewegingstiming als in positionering van ledematen. Hybride technieken combineren en concateneren beweging die gegenereerd is door verschillende animatie paradigmas, op zo'n manier dat zowel de natuurlijkheid en de controle verbeterd wordt.

Dit proefschrift introduceert zo'n hybride techniek: mixed dynamics. Mixed dynamics combineert de fysische natuurlijkheid van fysische simulatie met de controle van procedurele animatie. Het maakt gebruik van de notie dat het belang van fysische integriteit en strakke temporele synchronisatie vaak verschillend is voor verschillende lichaamsdelen. Bijvoorbeeld, voor een gesticulerend virtueel mens is temporele precisie is vooral belangrijk bij de synchronisatie tussen spraak en arm- en hoofdbeweging. Voor een gebalanceerde onderlichaamsbeweging is zulke precieze timing minder belangrijk; hier kan een fysisch realistisch balans controller gebruikt worden om een natuurlijke beweging te bereiken. Met mixed dynamics kan animatie uitgevoerd worden als een combinatie van procedurele gebaren en

fysische simulatie op verschillende lichaamsdelen. Hierbij worden de krachten die uitgeoefend worden door de procedureel aangestuurde lichaamsdelen terug gekoppeld op de fysisch aangestuurde lichaamsdelen. Hiermee wordt een animatie van het hele lichaam bereikt die op een natuurlijke manier aan de fysische wetten lijkt te voldoen en die intern coherent is (de beweging van de fysisch aangestuurde lichaamsdelen wordt beïnvloed door de beweging van de procedureel aangestuurde lichaamsdelen).

In traditionele dialoog systeem die gebruikt worden voor virtuele mensen werd interactie ontworpen met een 'zender/ontvanger' paradigma, waarin de gebruiker en de virtuele mens om de beurt informatie verzenden (encoderen) en ontvangen (decoderen). Zo'n interactie paradigma is niet afdoende om de rijkheid van mens-mens interactie (bijvoorbeeld in een conversatie) te vatten. Natuurlijke interactie vereist een continu interactie paradigma waarin de deelnemers de spraak en beweging van anderen continu observeren en continu, simultaan en derhalve overlappend in tijd handelen (spreken, gesticuleren). Zulke continue interactie vereist dat de perceptie van de virtuele mens snel is en dat de interpretatie van het gedrag van zijn gesprekspartner incrementeel uitgebreid en mogelijk aangepast kan worden. Om snelle observaties en continue aanpassing van de gedragsinterpretaties aan te kunnen moeten de multimodale output generatie modules van de virtuele mens op een flexibele manier gedrag kunnen genereren. Zulke flexibele generatie moet gedragselementen op een laat moment kunnen toevoegen, gedrag kunnen coördineren met voorspelde events in het gedrag van de gesprekspartner en moet gedrag kunnen aanpassen als het al gepland of aan het afspelen is. Dit proefschrift gaat over de specificatie en executie van zulk flexibel, multimodaal gedrag.

De Behavior Markup Language (BML) is de de facto standaard voor de synchronisatie van motor gedrag (inclusief spraak en gebaar) van virtuele mensen. BML wordt geïnterpreteerd door een *BML Realizer*. De BML Realizer voert het gespecificeerde gedrag uit op een virtueel mens. Applicaties waarin continue interactie met virtuele mensen nodig is hebben een aantal generieke specificatie vereisten. Aan een aantal van deze specificatie vereisten wordt door BML voldaan: BML specificeert de interne (dus binnen de virtuele mens) synchronisatie van gedrag en beschrijft de vorm van gedrag. Naast deze specificatie mechanismes vereist continue interactie specificatie mechanismes voor de interruptie van lopend gedrag, het aanpassen van de vorm van lopend gedrag (bijvoorbeeld: spreek luider) en de synchronisatie van gedrag aan voorspelde externe tijdsmomenten (bijvoorbeeld van de gesprekspartner). Dit proefschrift introduceert BML Twente (BML$^{\mathrm{T}}$), een taal die BML uitbreidt met de hierboven beschreven specificatie eigenschappen voor continue interactie. BML$^{\mathrm{T}}$ biedt dus een generieke interface voor een Realizer, waardoor continue interactie kan worden gerealiseerd.

"Elckerlyc" is ontworpen als een BML Realizer voor de generatie van multimodaal verbaal en non-verbaal gedrag voor virtuele mensen.[1] De belangrijkste ontwerp eigenschappen van Elckerlyc zijn dat (1) het specifiek is ontworpen voor

[1] "Elckerlyc" is de protagonist van het Nederlandse moraliteit spel met dezelfde naam, geschreven aan het einde van de middeleeuwen. The protagonist staat voor elk mens/iedereen, en beschrijft de tocht die gemaakt wordt aan het einde van het leven.

continue interactie, met strakke cordinatie tussen het gedrag van de virtuele mens en zijn gesprekspartner; (2) het een aanpasbare trade-off biedt tussen de controle en natuurlijkheid van verschillende animatie technieken (bijvoorbeeld procedurele lichaamsanimatie en fysische simulatie; MPEG-4 gezichtsanimatie en morphgebaseerde gezichtsanimatie); en (3) het is ontworpen als een *modulair en uitbreidbaar* systeem, dat kan worden uitbreid en aangepast zonder dat er invasieve modificaties in Elckerlyc zelf gemaakt hoeven worden.

Een BML Realizer is verantwoordelijk voor het uitvoeren van gedrag gespecificeerd in de BML blokken die er naartoe gestuurd worden, op zo'n manier dat er aan de tijdsconstraints die gespecificeerd worden in de BML blokken wordt voldaan. Realizer implementaties, waaronder Elckerlyc, gebruiken twee processen om dit voor elkaar te krijgen. Een planning proces is verantwoordelijk voor het creëren van een multimodaal gedragsplan. Een executie proces voert dit plan uit.

In de meeste BML Realizers resulteert de planning van BML in een rigide multimodaal realisatieplan, waarin de timing van het gedrag vast ligt. In Elckerlyc daarentegen, dicteren de continue interactie vereisten dat het multimodale gedragsplan regelmatig moet kunnen worden aangepast gedurende de executie van dit plan. Deze aanpassingen moeten op zo'n manier toegepast worden dat de tijdsconstraints de gespecificeerd waren in BML geldig blijven, en dat het resulterende gedrag biologisch uitvoerbaar is. Elckerlyc introduceert een *flexibele* multimodale plan representatie die plan aanpassingen toelaat, maar timing en natuurlijkheids constraints intact houdt.

Elckerlyc is de eerste BML Realizer die specifiek is ontworpen voor continue interactie. Het introduceert flexibele formalismen voor zowel de specificatie als de modificatie van lopend gedrag. Elckerlyc is het eerste multimodale virtuele mens systeem dat gebruik maat van real-time fysische simulatie en mixed dynamics. Hiermee wordt fysische coherente beweging over het hele lichaam gegenereerd. Deze natuurlijkheidseigenschap mist in virtuele mens systemen die alleen gebruik maken van procedurele animatie. Daarnaast biedt Elckerlyc een meer uitbreidbare en grondiger geteste architectuur dan bestaande BML Realizers. Andere Realizers implementeren alternatieve en uitgebreidere planning algoritmes, bieden motor gedrag op modaliteiten die niet aanwezig zijn in Elckerlyc (bijvoorbeeld blozen), of bieden gespecialiseerde gedragselementen (bijvoorbeeld lopen). Elckerlyc's uitbreidbaarheid zorgt ervoor dat zulk gespecialiseerd gedrag op nieuwe of bestaande modaliteiten op een gemakkelijke manier toegevoegd kan worden. Elckerlyc is ook ontworpen om het gebruik van nieuwe scheduling algoritmes toe te laten; de haalbaarheid van deze ontwerpeigenschap is nog niet bewezen.

Elckerlyc wordt gebruikt in een aantal virtuele mens-applicaties. De ontwerpeigenschappen van Elckerlyc zijn gemotiveerd, afgeregeld en gedemonstreerd door ervaringen van het gebruik van Elckerlyc in het 'veld'.

Contents

Chapter 1

Introduction

Researchers have always been fascinated with the application of the state-of-the-art technologies of their time to create artificial life, or, in particular, artificial humans [238]. Some of the first known examples of such artificial life designs are found in the Hellenistic world. Hero of Alexandria (10-70 AD) designed several automata or self operating machines, including a programmable cart and an owl-and-birds device featuring artificial birds that stop whistling as soon as an artificial owl looks at them. These automata were used for entertainment and to illustrate basic scientific principles, such as those of mechanics and pneumatics. In fifteenth-century Italy, automata made their appearance in theater plays and pageants. A famous example is Giovanni Fontana's she-devil, a mechanical devil that could move her facial features, tail, arms and wings and could shoot fire from her ears and mouth. Jacques de Vaucanson (1709-1782) pioneered the creation of what he called 'moving anatomies': machines that could simulate internal processes in living creatures such as digestion, respiration and blood circulation. His creations included a humanoid that was able to play the German flute using a simulated respiration system and the appropriate tongue and finger movements, and a mechanical duck containing over 400 moving parts, that could flap its wings, drink water, digest grain, and defecate.[1] Vaucanson commended his automata as appropriate instruments for instruction. He referred to the impression his three-dimensional mechanical objects could make on viewers, and to their anatomical accuracy and their unique ability to demonstrate life processes in real time [238].

The first virtual characters appeared in cartoons. Winsor McCay was one of the pioneers of cartoons. His 'Gertie the Dinosaur' cartoon (1914) features not only one of the first cartoons in which the character has an appealing personality, but also one of the first (staged) interactions of a human with a virtual character. McCay's interaction with Gertie consisted of him instructing her to do various tricks, throwing an apple to her (with Gertie catching an animated copy of it), and so on. The introduction of the computer allowed automation of the animation process and interaction with and between virtual humans. Early use of automation included automatic generation of the motion of virtual crash test dummies [306],

[1] The duck's digestive system was later found to be fake: the food was collected in one inner container, and the pre-stored feces was 'produced' from a second container [238].

automatic generation of locomotion [319] and 'programming' of animation using higher level descriptions (for instance by generating it from Labanotation [302]). Computer games often feature virtual humans that interact with each other and that can be interacted with. However, conversational interaction with and between game characters is typically completely scripted. Cassell et al. [50] pioneered automatic conversational interaction between autonomous virtual humans. Their virtual humans make use of automatically generated (using a dialog generation program) utterances. These utterances featured synchronized speech, facial expressions and hand gestures. Thórisson [282] contributed an architecture (Ymir) that was used to create Gandalf, one of the first virtual humans that could interact with a real human using speech and gesture. Gandalf not only generated speech and gesture, but could also perceive these communicative signals in humans. People talking with Gandalf wore a suit that tracked their upper body movement, an eye tracker that tracked their gaze, and a microphone that allowed Gandalf to hear their words and intonation. Gandalf's animation was displayed on a cartoon face and a disembodied hand. Ymir was one of the first architectures taking some aspects of continuous interaction into account, and, as such, its design remains influential in current virtual human platforms. A striking early example of the use of an interactive virtual human in a training application is Steve [235]. Steve is capable of teaching complex real-world tasks, that might be impractical to train on real equipment. His embodiment allows him to demonstrate actions, to use gaze and gesture to communicate and to guide the student in a virtual naval ship. Steve can also be used as a virtual team member to help a student practice his team tasks.

Nowadays, virtual humans have become very complex pieces of software. Building a state-of-the-art virtual human entails re-implementing several pieces of existing work. One of the current research directions in the interactive virtual human field deals with enabling more easy cooperation between research groups. To this end, the SAIBA initiative (consisting of several leading researchers in the interactive virtual human field) designed a framework that allows researchers to share components of virtual humans more easily [152]. Another current research direction deals with achieving the richness of human-human communication in communication with virtual humans. This entails designing virtual humans that allow continuous interpersonal coordination with their interlocutors [151].

1.1 Research Context

The research of this thesis was carried out within the Game research for Training and Entertainment (GATE) project[2], funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie). The GATE project aims to advance the state of the art in (serious) gaming, and to facilitate knowledge transfer to the industry. The work in this thesis was specifically done in the context of Work Package 2.1, which deals with the modeling and generation of motor behavior for virtual humans.

[2] http://gate.gameresearch.nl/

Some of the work in this thesis was done in the context of the Knowledge Transfer Project 'Computer Animation for Social Signals and Interactive Behaviors', within the GATE project mentioned above. The goal of this project is to transfer the knowledge of the Human Media Interaction group on multi modal virtual human behavior generation to our industry partner Re-lion.

The focus of my work within those projects is on the output generation and specification of the behavior (including speech, body motion, facial motion) of *interactive virtual humans*.[3]

1.2 Relevance

Interactive virtual humans are used in many educational and entertainment settings: serious gaming, interactive information kiosks, kinetic and social training, tour guides, storytelling entertainment, tutoring, interactive virtual dancers, entertaining games, motivational coaches, and so on. Virtual humans have an embodiment that inhabits a virtual environment. This gives a virtual human interactive capabilities that go beyond written text or video: a virtual human can guide a human through the virtual world and is able to demonstrate actions in this world.

In addition to their use in education and entertainment, virtual humans provide valuable research tools. Social psychologists can study theories of communication by systematically modifying the behavior of a virtual human. Using virtual humans and virtual environments rather than human actors and custom built mock-up environments in social psychology experiments allows more experimental control and better reproducibility [32]. Interactive virtual humans can also be used to *simulate* formal models of, for example, human conversation. Through such simulations, our understanding of human conversation can be improved [47]. They highlight gaps in these formal models and thus show where further modeling or refinement is required.

1.3 Research Goals and Contributions

1.3.1 Enabling Collaboration and Competition in Virtual Human Design

Designing a virtual human is a multi-disciplinary effort, requiring expertise in many research areas, including computer animation, perception, cognitive modeling, emotions and personality, natural language processing, speech recognition, speech synthesis, nonverbal communication [98]. Research groups have realized that 'the scope of building a complete virtual human is too vast for any one research group'

[3] I use the term interactive virtual humans instead of Embodied Conversational Agents [51] in this thesis, because the virtual characters this thesis deals with are human-like and the interaction with them is not necessarily in the form of a conversation.

[141]. Modular architectures and interface standards will allow researchers in different areas to reuse each other's work and thus allow easier collaboration between researchers in different research groups [98]. Interface standards also promote healthy competition between research groups who create modules that implement them, since they allow an easy comparison between such modules. The SAIBA initiative proposes an architecture for virtual humans [152] that provides such a modular design with standardized interfaces. The Human Media Interaction group has joined the SAIBA initiative and contributed towards the interface (the Behavior Markup Language, BML) for one of its modules: the Behavior Realizer. Such a Behavior Realizer provides an interface to steer the coordinated motor behavior of a virtual human (e.g. speech, gesture).

This thesis contributes an implementation of such a Behavior Realizer. I aim to promote, measure, test and maintain the SAIBA compliance of Behavior Realizers. To this end, I contribute the automatic testing framework RealizerTester that can test adherence to the SAIBA interface for *any* Behavior Realizer. The modular design of my Realizer enables collaboration opportunities beyond those offered by implementing the SAIBA interface. It makes it possible for other research groups to easily connect it to their own rendering environment or virtual human and to add specific modularities (e.g. to control a robot), without having to make invasive modifications to the Realizer itself.

1.3.2 Designing a Virtual Human that Allows Continuous Interaction

Traditionally, interaction with virtual humans was designed using 'transmitter/receiver' interaction paradigms, in which the virtual human and the human interacting with it take turns to transmit (encode) and receive (decode) meaning carrying messages that travel across channels between them. Such an interaction model is not sufficient to capture the richness of human-human interaction (including conversation). Natural interaction requires a *continuous* interaction paradigm, where actors perceive acts and speech of others continuously, and where actors can act continuously, simultaneously and therefore overlapping in time. I aim to design a virtual human that allows such continuous interaction. A design for continuous interaction should however not come at the cost of the modularity provided by the SAIBA framework.

This thesis describes a view of the SAIBA framework that allows continuous interaction. I describe the requirements of continuous interaction and contribute the interface language elements — in BML Twente (BML^T), an extension of BML — that allow it. I also contribute the Behavior Realizer "Elckerlyc", specifically designed to allow the execution of behavior of a virtual human in applications that require continuous interaction with a human interlocutor.

1.3.3 Leveraging Computer Animation Knowledge for Interactive Virtual Human Applications

In typical interactive virtual human applications, the movement of the virtual human consists solely of animations on the head and arms, synchronized with speech. Gesture movement is typically generated by intricate procedural models that implement biological rules for arm movement [155], or provide emotional parameterization [104, 111] and provide tight synchronization to speech. However, the movement of the rest of the body is either completely omitted, provided by noise uncorrelated to the arm and head movement, or set by some predefined idle animation [104, 111, 155, 235, 282]. Treating gesture as a movement that is localized in the limbs results in motions that lack impact and are perceived as being robotic [58]. Many state-of-the-art computer animation techniques achieve more natural movement, often at the cost of movement control. I aim to leverage the knowledge of computer animation for researchers in interactive virtual human applications.

To this end, this thesis contributes a thorough overview of real-time animations techniques that can be used for the generation of natural human motion, with a focus on the different trade-offs between naturalness and movement control offered by these techniques. It also contributes mixed dynamics: a novel hybrid animation technique that can combine different kinds of animation paradigms, allowing the combination of traditional procedural gesture animation or keyframe animation with physical simulation, both in sequence and in parallel on different body parts. This allows one to combine the control of procedural (gesture) animation, with the naturalness of physical simulation.

1.4 Outline of this Thesis

Figure 1.1 provides a graphical outline of the work in this thesis in relation to the SAIBA architecture. The SAIBA architecture models behavior generation in three planning processes: Intent Planning, resulting in a script in the Functional Markup Language (FML); Behavior Planning, resulting in a script in the Behavior Markup Language (BML) and Behavior Realization of the BML script. In this thesis, I split Behavior Realization into *scheduling*, resulting in a Motor Plan; and the *execution* of this Motor Plan, resulting in control primitives (e.g. joint rotations, audio) that are used to steer embodiment of a virtual human. To allow continuous interaction, it is important that an ongoing Motor Plan is flexible and can be modified on the fly. The Multimodal Behavior Plan provides an abstraction of the Motor Plan that is used to apply such modifications.

Chapter 2 provides an overview of computer animation techniques that can be used to execute Animation Plans (Motor Plans for animation) and provides an overview of the naturalness and control tradeoffs made by these different techniques. Chapter 3 discusses mixed dynamics: a system to simultaneously execute animation expressed in kinematic PlanUnits and PlanUnits that make use of physical simulation. Chapter 4 discusses the Motor Plan. It provides a brief overview of the coor-

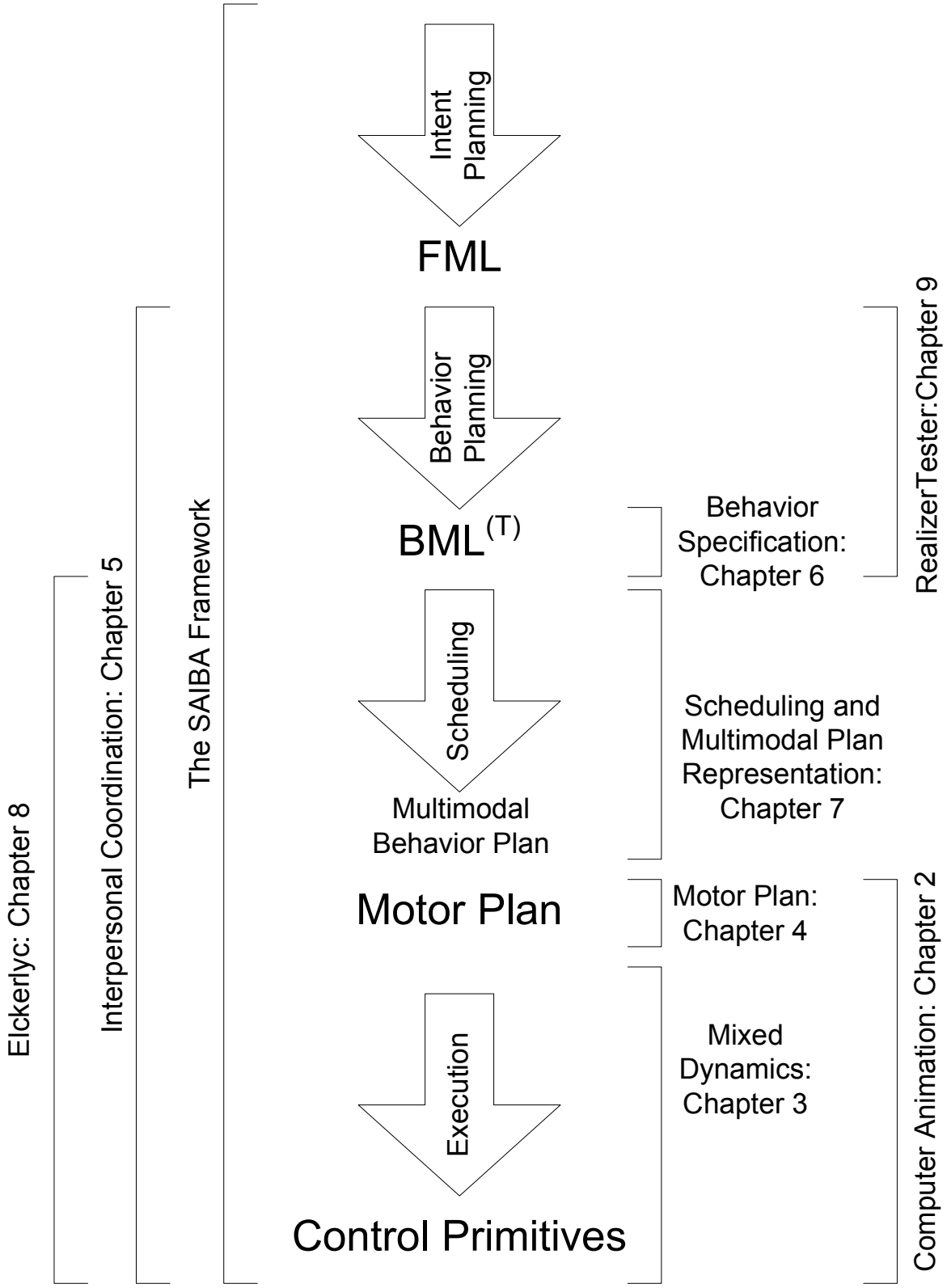


Figure 1.1: Thesis overview.

dination between the PlanUnits of the Motor Plan, provides an interface for flexible PlanUnits, and discusses the implementation of several PlanUnits used for animation. Chapter 5 discusses the interpersonal coordination of the behavior of humans, why it is important to model this in virtual humans, and how interpersonal coordination can be achieved at several levels in the SAIBA architecture. Chapter 6 deals with the specification of multimodal behavior for virtual humans. It describes how coordination between PlanUnits in the Motor Plan is specified through BML and provides a BML extension (BML^{T}) that allows the specification of the behavior of a virtual human in applications that require continuous interaction. Chapter 7 deals with the scheduling of BML into a Motor Plan. It introduces a flexible multimodal plan representation that allows one to modify an ongoing Motor Plan on the fly, while maintaining the constraints posed upon it in the BML script(s) that created it. Chapter 8 introduces Elckerlyc, a modular and flexible BML Realizer that can schedule and execute behavior plans specified in BML$^{(T)}$. Chapter 9 discusses some of my efforts towards measuring, testing and promoting the compliance of BML Realizers to the BML standard. It contributes RealizerTester, a generic framework to test *any* BML Realizer. Chapter 10 demonstrates how Elckerlyc's design features worked out in practice and shows how one can build virtual human applications using Elckerlyc. I wrap up this thesis in Chapter 11 and end it (in Chapter 12) by discussing how Elckerlyc's contributions on the coordination of the form and timing of the behavior with an interlocutor could be combined with both work on the coordination of content and form and work on continuous and incremental input processing.

Chapter 2

Real-Time Computer Animation: a Review[†]

Virtual environments inhabited by virtual humans are now commonplace in many applications, particularly in (serious) games. The animation of such virtual humans should operate in real-time to allow interaction with the surroundings and other (virtual) humans. For such interactions, detailed *control* over motion is crucial. Furthermore, the motion of virtual humans should *look* realistic. I use the term *naturalness* for such perceived realism.

Many techniques achieve real-time animation. These techniques differ in the trade-off they offer between the control that can be exerted over the motion, the motion naturalness, and the required calculation time. Choosing the right technique depends on the requirements of the application it is used in. This chapter provides an overview of real-time animation techniques that can potentially be used in interactive virtual human applications. It provides a short summary of each technique, and focuses on the trade-offs made.

First, I discuss models of the virtual human's body that are steered by animation (Section 2.1). In Section 2.2, I classify animation techniques that are used to generate motion primitives and discuss their strengths and weaknesses. Section 2.3 discusses how to parameterize, combine (on different body parts) and concatenate motion generated by these techniques to gain control. In Section 2.4, I elaborate on several aspects of naturalness and I discuss how the naturalness of the motion of a virtual human can be evaluated. I conclude (in Section 2.5) by discussing the power of combinations of animation paradigms to enhance both naturalness and control.

[†]This chapter is largely based upon the article:
H. van Welbergen, B.J.H. van Basten, A. Egges, Z.M. Ruttkay and M.H. Overmars. Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control, Computer Graphics Forum, 29(8):2530-2554, 2010

2.1 Modeling the Virtual Human

Animation steers the body of a virtual human. In this section it will be shown how the body of a virtual human is modeled as a skeleton, an articulated set of rigid bodies and a biological system.

2.1.1 Skeletal Model of the virtual human

Virtual humans are visually mostly represented by polyhedral models or *meshes*. Animating all these polygons individually can be very tedious, therefore it is common to work with the underlying *skeleton* instead. A skeleton is an articulated structure: a hierarchy of segments connected by joints. A *pose* of a virtual human is set by rotating the joints of the skeleton. How the skeleton deforms the mesh is beyond the scope of this thesis, I refer the interested reader to [184].

Every joint has several degrees of freedom or *DoFs*. The DoFs are the parameters that define a configuration of a joint. For example, the knee joint has only one DoF, while a shoulder joint has three. The global translation of the skeleton is represented by the translation of the root joint. The pose of a skeleton with n rotational DoFs can therefore be described by an $n + 3$ dimensional vector $\mathbf{q}$. For an overview of rotation representations I refer the reader to the work of Lee [167].

Standardizing the skeleton topology improves re-usability of motions. Motions created for one virtual human can be transfered to another virtual human more easily. The H-anim standard [119] provides a complete set of standardized joint names and their topology, that specifies their resting position and how they are connected.

2.1.2 Physical Model of the Virtual Human

In physical simulation, the body of the virtual human is typically modeled as a system of rigid bodies, connected by joints. Each of these rigid bodies has its own mass and an inertia tensor that describes the mass distribution. Movement is generated by manipulating joint torques.

Most physical animation systems assume a uniform density for each rigid body. Given such an uniform density, the mass, center of mass and inertia tensor can be calculated via the volume of the mesh that corresponds to the rigid body (see [195]). Realistic values for the density of the rigid bodies can be obtained from the biomechanics literature [307].

To allow for collision detection and collision response, a geometric representation of the rigid bodies is needed. The mesh of the virtual human can be used for this representation. However, collision detection between arbitrary polygonal shapes is time consuming. Computational efficiency can be gained at the cost of some physical realism by approximating the collision shape of rigid bodies by basic shapes such as capsules, boxes or cylinders.

2.1.3 Biomechanical/Neurophysical Models of the Virtual Human

Our movements are coordinated by the central nervous system (CNS). It uses input from sensors to steer our muscles. These sensors, muscles and the motor control exerted by the CNS have, to some extent, been modeled in computer animation.

2.1.3.1 Sensors

Motor control needs information on the state of the virtual human. This information is readily available from the representation of the virtual world. Sensors used in computer animation therefore do not necessarily need to correspond to the sensors found in humans, but merely represent a convenient higher level presentation of virtual human state information that can be shared between different motion controllers [73]. Examples of information obtained by such sensors are the center of mass (CoM) of the virtual human, contact (are the feet or other body parts in contact with the ground?), the location of the support polygon (the convex hull of body parts touching the ground), and the zero moment point (ZMP). The ZMP is the point on the ground plane where the moment of the ground reaction forces is zero. In all physically realistic motion with ground contact, the ZMP is inside the support polygon. If the ZMP is outside the support polygon, the virtual human is perceived as being out of balance [265].

2.1.3.2 Modeling Muscles

Over 600 muscles can apply forces to our bones by contracting. One muscle can cover multiple joints (e.g. in the hamstring and muscles in the fingers). In real-time physical simulation methods, muscles are typically modeled in a simpler manner: as motors that apply torques at the joints in an articulated rigid body system (as set up by the physical model of the human, see Section 2.1.2). Such a model provides control in real-time and has a biomechanical basis: it is hypothesized that the CNS exerts control at a joint and joint synergy level [307]. To determine the torque applied by these motors, muscles are often modeled as a system of springs (representing elastic tendons) and dampers that cause viscous friction [307]. In real-time animation, such spring and damper systems are often designed using Proportional Derivative(PD) controllers or variants thereof (see Section 2.2.3.2,2.2.3.2). Joint rotation limits and maximum joint strength can be obtained from the human factors literature (see for example: [145, 310]).

2.1.3.3 Models for Motor Control

Motor control is the process that steers the muscles in such a way that desired movement results. In many cases robotic systems can rely on control based directly on internal feedback (e.g. using joint angle sensors). Feedback delays in humans are large (150-250 ms for visual feedback on arm movement), so they cannot achieve

accurate fast movement using solely feedback control [132]. According to Schmidt [254] people construct parameterized *General Motor Programs* (GMPs) that govern specific classes of movement. Different movements within each class are produced by varying the parameter values. Humans learn the relation between parameter values and movement 'outcome' by practicing a task in a great variety of situations.

According to the *equilibrium point hypothesis*, joint torque paths are not explicitly programmed, but emerge from the dynamic properties of the biomechanical system. In this model, the spring-like properties of muscles in, for example the arm, are used to automatically guide the hand to an equilibrium point. Movement is achieved by a succession of equilibrium points along a trajectory [76]. Feedback control (see Section 2.2.3.2), GMPs (explicitly in [155, 319], implicitly in Sections 2.2.2, 2.2.1.2) and equilibrium point control (see Section 2.2.3.2) are all used in computer animation.

The GMP theory is supported by invariant features that are observed in motion. Gibet et al. [86] give an overview of some of such invariant features, including Fitts' law, the two-third power law and the general smoothness of arm movement. Fitts' law states that the movement time for rapid aimed movement is a logarithmic function of the movement distance divided by the target size [77]. The two-third power law [299] models the relation between the angular velocity and the curvature of a hand trajectory. Movement smoothness has been modeled as a minimization of the mean square of hand jerk (derivative of acceleration) [78] or the minimization of the change of torque on the joints executing the motion [295]. Harris and Wolpert [103] provide a generalized principle that explains these invariants by considering noise in neural control. The motor neurons that control muscles are noisy. The variability in muscle output increases with the strength of the command. For maximum accuracy it is therefore desirable to keep the strength of motor commands low during the whole movement trajectory, thus producing smooth movement. Faster movement requires stronger motor commands, thus higher variability which leads to reduced precision. In computer animation, movement invariants have been used both in motion synthesis models (for example: [87, 155]) and as evaluation criteria for the naturalness of animation (see Section 2.4.5.2). The notion of signal dependent noise has been exploited in the generation of motion variability (see Section 2.4.4.3).

2.2 Animation Techniques

A motion primitive is a continuous function that maps time to the DoF of a skeleton. Animation techniques create *motion primitives* from *motion spaces* on the basis of *animation parameter* values (see Figure 2.1). A motion space is a (continuous) collection of motions that can be produced by a technique. A motion primitive is an element of such a motion space. Motion primitives can define motion for the full body of a virtual human or on a subset of the joints of the virtual human. The motion primitives in a specific motion space typically have a certain semantic function (for example: walk cycles, beat gestures, left hand uppercuts). The animation

parameters needed to create motion primitives differ per technique. Note that animation parameters are not necessarily intuitive parameters to control motion, but merely the parameters a specific animation technique requires to create a motion primitive. I discuss how to map more intuitive *control parameters* into animation parameters in Section 2.3.1.

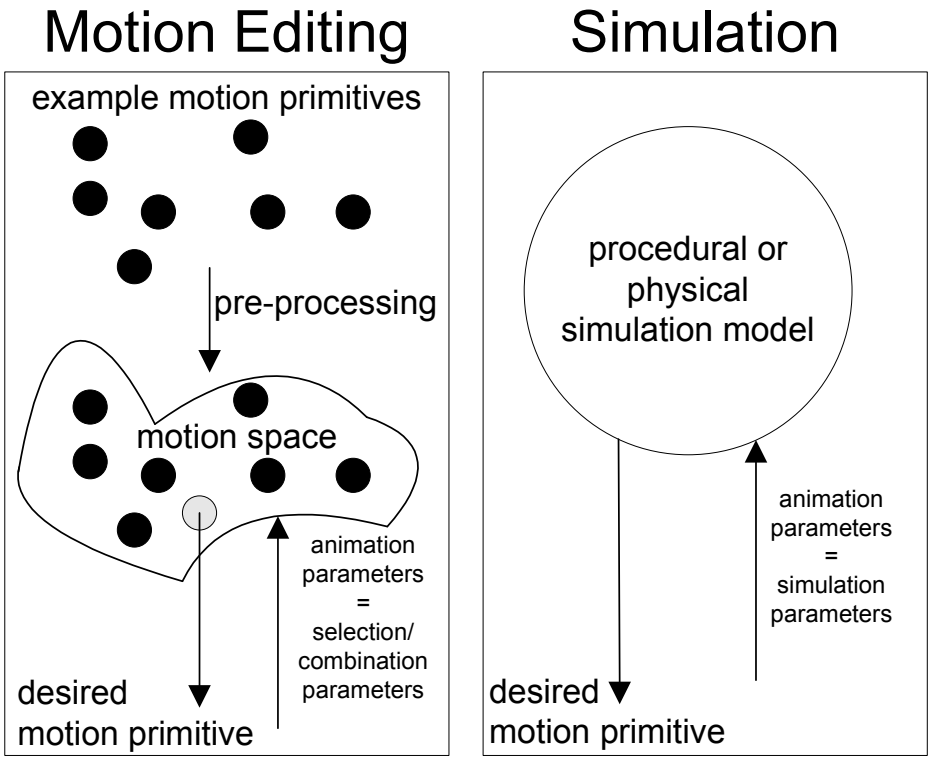


Figure 2.1: Motion primitives, motion spaces and animation parameters in motion editing and in simulation.

In this thesis, animation techniques are classified by the mechanism they use to create motion spaces (see Figures 2.1 and 2.2). *Motion editing* techniques generate motion primitives within a motion space spanned by one or more specific example motion primitives. In *simulation* techniques, the motion space contains all motion primitives that can be created using a parameterized physical or procedural model. Animation parameters in simulation techniques are the parameters used in the simulation model. In Sections 2.2.1, 2.2.2 and 2.2.3, I briefly discuss the inner working of each technique and discuss the nature of its animation parameters and motion spaces produced by the technique. Figure 2.2 provides a summary of the latter. Section 2.2.4 discusses the strengths and weaknesses of each technique in terms of naturalness and control and gives an overview of application domains in which each of the techniques is typically used.

2.2.1 Motion Editing

Motion editing techniques generate motion primitives within a motion space spanned by one or more specific example motion primitives. Often, this motion space

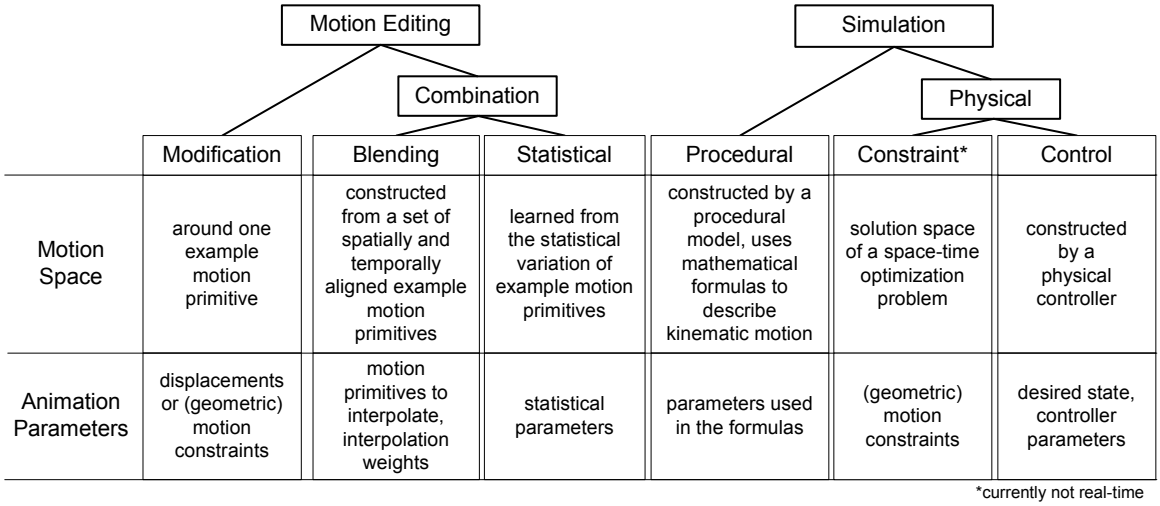


Figure 2.2: Classification of animation techniques and an overview of their animation parameters and motion spaces.

is explicitly constructed in a pre-processing stage. The example primitives originate from motion captured movement of actors, or are created by hand by an animator. I define *motion modification methods* as methods that generate new motion primitives by applying modifications to a *single* example motion primitive. *Combination techniques* create motion primitives using a database of *multiple* example primitives.

2.2.1.1 Motion Modification

Since a motion primitive is a continuous function that maps time to the DoF of a skeleton, this value of a DoF over time can be considered as a *signal*. Therefore many techniques from the field of signal processing can be applied to create a motion space around an example motion primitive. Bruderlin and Williams [41] consider some motion editing problems as signal processing problems. One of the signal processing techniques they use is *displacement mapping*. With this technique it is possible to make local modifications to the signal while maintaining continuity and preserving the global shape of the signal. This is done by specifying some additional keyframes, or having them determined by inverse kinematics (IK, see Appendix A for an overview of techniques), within an example motion primitive. From these keyframes, a displacement map can be calculated that encapsulates the desired displacement (offset) of the signal. Splines can be used to calculate the inbetween displacements. The displacement map then yields a displacement for every frame, which is automatically added to the original signal. Satisfying constraints at key frames does not guarantee constraint enforcement at the 'inbetweens' (the frames between the keyframes). Alternatively, a constraint can be enforced at *every* frame on which it is desired, as proposed by Lee and Shin [170]. To make sure the resulting motion is smooth and propagated through non-constrained frames, it is 'filtered' using a hierarchy of B-splines. Gleicher [91] calls the family of solutions that uses such an approach 'Per Frame Inverse Kinematics + Filtering' (PFIK+F).

An alternative approach by Gleicher [90] is to pose the constraint specification as a numerical constraint optimization problem: an objective function measuring the distance between the example motion primitive and the resulting motion is minimized subject to any constraint that can be specified as a function of the vector of DoF $\mathbf{q}$. To allow real-time execution of this optimization, an efficient objective function is chosen and the constraints are only enforced at key frames. The geometric constraints that can be solved with PFIK+F are a subset of those that can be solved using the optimization approach. Optimization can add (among many other things) constraints for a region an end effector must stay in, fixed distances between end-effectors or inter-frame constraints. This flexibility comes at a cost: it is not ensured that the constraints are met at the inbetweens and the solution time of the optimization process is less predictable than that of a PFIK+F approach. I refer the reader to [91] for a more thorough comparison of the two methods.

2.2.1.2 Blending

Blending [305] creates a motion primitive by interpolating a family of similar example motion primitives (for example: a family of reaching motion primitives, walking motion primitives, etc.). The animation parameters are interpolation weights and a selection of the example motion primitives to interpolate. The interpolation does not need to take place in the Euler space, but can also be done in, for example, the principal component [120] or Fourier [296] domain. In general, one can only interpolate between poses that "resemble" each other. When this is not the case, visual artifacts such as foot skating may appear. A distance metric quantifies the resemblance between poses. Van Basten and Egges [18] present an overview and comparison of various distance metrics.

The blend motion space is created by pre-processing "similar" example motion primitives, typically such that they correspond in time (especially at key events such as foot plants) and space (e.g. root rotation and position). The process of time-aligning corresponding phases in motion primitives, is called time warping [41]. Kovar and Gleicher [156] present an integrated method called registration curves to automatically determine the time, space and constraint correspondences between a set of motion primitives and provide a literature overview of earlier methods used for this.

2.2.1.3 Statistical models

Statistical methods create a motion space using statistical models learned from the statistical variation of example motion primitives. Several statistical models can be used, including Hidden Markov Models (HMM)[38], Linear Dynamic Systems [173], Scaled Gaussian Process Latent Variable Models (SGPLMVM) [100], Principle Component Analysis (PCA) [69], or variogram functions [199].

2.2.2 Procedural Simulation

Procedural simulation uses parameterized mathematical formulas to create motion primitives. The parameters of such formulas are the animation parameters. The formulas can describe joint rotation directly (as done in [217]), or describe the movement path of end effectors (such as hands) through space. The latter is typically used to design procedural models that create gesture motion primitives (see for example [58, 104, 155, 207]).

2.2.3 Physical Simulation

A physical simulation model applies torques on the joints of the virtual human, on the basis of animation parameters. The resulting motion primitive is calculated using forward dynamics (see Appendix A).

2.2.3.1 Constraint Control Methods

Constraint Control Methods use (geometric) constraints as animation parameters. There are typically many possible muscle torque paths that achieve the constraints. An objective function can be introduced to specify a certain preference for solutions. Typically, the objective functions are biomechanically based: minimize the expended energy, minimize end effector jerk, or use a weighted combination of those two. The constraint control problem can be stated as a non-linear optimization problem [308]. Several techniques have been proposed to speed up the calculation process of the optimization (for example: [74, 174]), typically at the cost of some physical realism. Even with those speedups, constraint based control methods are currently not a feasible option for real-time animation.

2.2.3.2 Physical Simulation using Controllers

A physical controller and the physical system it controls (the physical body of a virtual human) together form a control system [149]. The input to the controller is the desired value of the system's state. This desired state is part of the animation parameter set. The output is a set of joint torques that, when applied to the system, guides its variables towards their desired values. The controller can make use of static physical properties (such as mass, or inertia) of the physical body performing the motion. Such a control system can, to a certain extent, cope with external perturbation, in the form of impulses, forces or torques exerted on the body. The goal of the controller is to minimize the discrepancy between the actual and desired state. In addition to the forces and torques set by the controller, gravity and ground contact forces, and forces and torques caused by external perturbations are also applied to the physical body. The body is then moved using forward dynamics. The new state of the body is fed back into the controller.

Proportional Derivative (PD) Control is an easy to implement and frequently used control method (for example in [6, 73, 115, 312, 323]). The output torque of the PD-controller is proportional to the difference in position and velocity between the desired state and the actual state:

$$\tau = k_p(x_d - x) + k_d(\dot{x_d} - \dot{x}) \tag{2.1}$$

in which x_d is the desired state, x is the actual state and k_p and k_d are the proportional and derivative gains. Note that the system reacts similarly to a springer-damper system, with spring gain k_p and damper gain k_d. Typically x_d is a desired DoF value, but other state variables are used in more complex PD-controllers (such as CoM position in balancing [312]). The animation parameters that have to be used to create a motion primitive are k_p, k_d, x_d and $\dot{x_d}$. Finding appropriate values for k_p and k_d that result in achieving x_d and $\dot{x_d}$ is a manual trial-and-error process. They depend on characteristics of both the system and the motion.

Antagonist Control Neff and Fiume [202] use a slightly different formulation of the PD-control equation, that has more intuitive animation parameters, but the same error response. It is based on agonist and antagonist muscle groups around joints, that are modeled as springs:

$$\tau = k_{p_L}(\theta_L - \theta) + k_{p_H}(\theta_H - \theta) - k_d\dot{\theta} \tag{2.2}$$

in which animation parameters θ_L and θ_H are the spring set points, which serve respectively as desired lower and upper limits for the joint rotation θ. τ is the output torque. k_{p_L} and k_{p_H} are the spring gains. Equilibrium point control (see Section 2.1.3.3) is used to calculate k_{p_L} and k_{p_H}, given the provided stiffness and external forces (typically gravity). Movement is achieved by gradually moving the equilibrium position.

Local Optimization PD-controllers typically do not generalize well beyond the specific physical body, environment and contact conditions they were designed for. Controllers using local quadratic optimization provide better generalization. They optimize the control objectives for the current frame, subject to certain constraints (e.g. the physical equations of motion). Unlike constraint control methods (see Section 2.2.3.1), these controllers cannot anticipate the long term minimization of their objective, given constraints at certain time frames, but do allow real-time execution. The computation cost of local optimization is far higher than the computation cost of PD or similar controllers.

Stewart and Cremer [277] introduce a custom physics simulator that can optimize objectives (which are required to be second order derivatives of system variables), subject to the physical equations of motion and optionally specific constraints that can be added on the fly. Abe et al. [2] extend this work by designing controllers that optimize objectives, subject not only to the physical equations of motion, but also to contact and friction dynamics and maximum joint strengths. Their system is

designed to work with any physics simulator. The objectives regulate the values of certain kinematic quantities $f(\mathbf{q})$, by minimizing the difference between the desired acceleration of f and its current acceleration. Abe et al. [2] show some strategies to find the desired acceleration of $f(\mathbf{q})$ for balancing controllers and controllers that track a prescribed joint rotation trajectory.

Automatic Controller Generation Searching techniques or evolution-based machine learning techniques have been employed to automatically generate controllers that map sensor inputs (joint angles, ground touch) to joint torques, in such a way that an animation parameter (distance traveled, energy expended, distance from stylized reference pose) is optimized (see for example [263, 270]). Using such techniques, locomotion controllers for simple creatures with few DoFs can be created. However, so far automatic controller generation techniques have not scaled up to provide natural motion for full-sized virtual humans.

Physical Controllers Toolkits The Dynamic Animation and Control Environment [260] provides researchers with an open, common platform to test out and design physical controllers using scripting. NaturalMotion's Endorphin [200] is a commercial animation system that provides a predefined set of controllers. It offers animation authoring through controller parameterization, controller combination, physical constraint handling (e.g. lock hands to a bar for a 'hang on bar' motion) and several ways to integrate motion capture with physical simulation. NaturalMotion offers the Euphoria toolkit to handle such functionality in real time so that it integrates with a game engine. Details on how Naturalmotion software handles this functionality (as far as disclosed) are discussed in the appropriate sections.

2.2.4 Strengths and Weaknesses of Different Animation Techniques

Motion editing techniques retain the naturalness and detail of recorded example motion primitives or motion primitives generated by skilled artists. However, motion editing techniques produce natural motion only when the modifications to the example motion primitives are small. Techniques that make use of multiple example motion primitives retain naturalness over larger modifications than techniques that use a single example motion primitive [96]. However, both blending and statistical techniques suffer from the curse of dimensionality: in practice the number of required example motion primitives grows exponentially with the number of animation parameters [92]. Furthermore, motion editing techniques do not provide physical interaction with the environment and motion editing can invalidate the physical correctness of motion (see Section 2.4.1). Motion editing is useful for creating animation in advance for non-interactive applications (such as films). For other domains, such as games, naturalness and controllability can only be assured by using a huge database of example motion primitives.

Physical simulation provides physically realistic motion and (physical) interaction with the environment. Physical controllers can robustly retain or achieve animation parameters under the influence of external perturbation. This robustness comes with a disadvantage: precise timing and limb positioning using physical controllers is an open problem (see Section 2.3.1.5). While physical simulation provides physically correct motion, this alone is often not enough for motion to be natural. Therefore, physical simulation is mainly used to generate human motion that is physically constrained and in which interaction with the environment is important, such as motion by athletes (for example in [115, 313]), stunts by stuntmen [73], or falling motions (for example [187, 262, 312]).

Procedural animation offers precise timing and limb positioning and can easily make use of a large number of parameters. However, it is hard to incorporate movement details such as those found in example motion primitives into the mathematical formulas that create motion. Furthermore, to maintain physical naturalness, it has to be explicitly authored in the procedural model for all possible parameter instances. Expressive motion, as used in talking and gesturing virtual humans, requires many control parameters and precise timing to other modalities, such as speech. It is therefore typically the domain of procedural animation techniques such as [58, 104, 155, 207, 217, 218].

The qualities of motion editing and motion simulation techniques can potentially be combined by taking into account which of the qualities is needed in a certain situation, or by determining which quality is needed on which body part. For example, a virtual human can be steered by motion editing until a physical interaction with the environment is needed, which will then be handled by physical simulation, or the flexibility and precision of procedural motion can be used to generate arm gestures on a virtual human which retains balance in a physically realistic manner using a balance controller on the lower body. Throughout the remaining sections, I will show several examples of such combinations that enhance naturalness and/or control, as I discuss the control and naturalness provided by different animation techniques.

2.3 Control

Animation involves the creation of *animation plans* that typically span multiple motion spaces and are executed by multiple motion primitives. To be able to deal with interactive and changing environments, such plans need to be constructed and adapted in real time. Control enables the expression and adaptation of such plans by means of parameterization, combination and concatenation.

Parameterization (see Figure 2.3) is the process of selecting *animation parameter* values (such as blend weights, stiffness gains, Principal Component values, etc.) that, when provided to an animation technique, create a motion primitive that satisfies some *control parameters* (for example: create a gesture motion primitive that exhibits a certain tension and amplitude).

Concatenation (see Figure 2.4) deals with the generation of a sequences of mo-

tion primitives, to form a natural motion that satisfies certain control parameters. The motion primitives can be generated by different techniques (or the same technique that is initialized differently). For example: using a walk controller and a blending technique that uses preprocessed sit down motion primitives as its input, a sequence of motion primitives can be generated to achieve a 'walk to the chair and sit on it' motion.

Rather than explicitly constructing new motion primitives for each combination of motions acting on a separate body part, different motion primitives, possibly constructed by different animation techniques, can be *combined* (see Figure 2.5) in such a way that a coherent whole body motion results. For example: a walk cycle motion primitive and a gaze motion primitive can be combined to allow a virtual human to walk and gaze at the same time.

Controllability is determined by various aspects. *Responsiveness* determines how fast a desired change in the motion plan is achieved. For example, how fast does an animated soccer player respond to a gamer pressing the shoot button? *Precision* is the accuracy with which control parameters (such as end effector position or timing constraints) are achieved. *Coverage* deals with how much of the control parameter space is covered (for example: what positions can be kicked within a kick motion space). I define *expressiveness* as the number of control parameters that can be used in the motion plan. *Intuitiveness* deals with how intuitive the control parameters that can be used in the specification of the motion plan are for human motion authors.

2.3.1 Parameterization

Parameterization deals with selecting the animation parameters, that, when provided to an animation technique, create a motion primitive with some desired control parameter values. One common control parameter is a pose constraint (for example: requiring the hand to be at a certain location) at a desired time. Other more abstract parameterizations deal with control parameters such as emotion or physical state (such as tiredness). Some animation techniques provide intuitive animation parameters that can be expressed directly as control parameters (for example: geometric constraints in motion modification). Other techniques (for example blending) do not provide intuitive animation parameters. For such techniques and for the more abstract parameterizations mentioned above, some mapping of control parameters to animation parameters is needed (see Figure 2.3). If multiple desired control parameter values are specified, it might not be possible to satisfy them all. Several parameterization methods therefore include strategies to deal with conflicting control parameter values.

2.3.1.1 Parameterization in Procedural Motion

In procedural animation, the control parameters can directly be expressed in terms of parameters of the motion functions (and thus animation parameters). Pose constraints are typically satisfied by setting animation parameters that specify end effector positions or joint rotations. Authoring procedural motions requires specifying

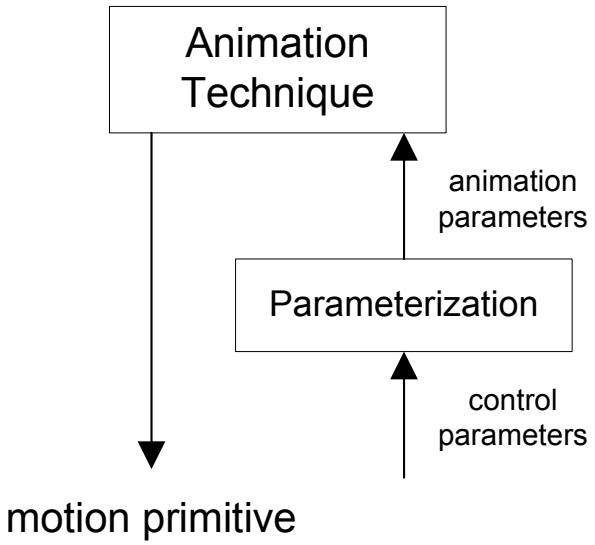


Figure 2.3: Parameterization maps control parameters to animation parameters to create a motion primitive that satisfies control parameter values.

how each parameter influences the motion. For control parameters such as emotion or physical state, this is not a very intuitive process. Therefore, such control parameters are typically mapped to animation parameters instead. This mapping can result in parameter conflicts if control parameter values select different values for the same animation parameter.

Neff and Fiume [203], design a hierarchical framework for procedural motion and provide a generic parameter mapping framework. Lower level control parameters specify the motion on a single joint or group of joints (called an action in [203]). Higher level control parameters are mapped to animation parameters through a script created by an animator. Motion primitives are created using various, possibly conflicting, low level and high level control parameters. Therefore, several mechanisms are in place to handle conflict resolution: low level control parameters (placed on a single DoF, rather than on the whole body) take precedence over high level control parameters, which take precedence over the default values defined in a 'Sketch' (a model of the virtual human's style in [203]; see Section 2.4.3.2).

Chi et al. [58] claim that Effort and Shape parameters from Laban Movement Analysis (LMA) not only provide means to parameterize gesture, but are essential features of a gesture. Shape involves the changing forms that the body makes in space. Effort describes dynamic qualities of movement, such as weight (light, for example dabbing paint on a canvas vs. strong, for example punching someone in the face in a boxing match) and flow (uncontrolled, for example shaking off water vs. controlled, for example carefully carrying a cup of hot tea). Their work provides a computational framework that maps Effort and Shape control parameters to animation parameters that guide arm and torso movement. The arm movement is specified by end effector key locations. Shape parameters influence the position of the hand in space on those key locations. Effort parameters influence the path and timing of the movement toward the end effector location. In later work, Badler

et al. [13] achieve emotional parameterization by mapping emotion to LMA control parameters.

Hartmann et al. [104], use a smaller but quite similar set of control parameters. From a literature review they conclude that six control parameters (activation, spatial extent, temporality, fluidity, power and repetition) are sufficient to specify gesture expressivity. The control parameter selection is based on what humans can observe and reliably recognize. In their system, gestures are generated by Kochanek-Bartels splines [146] defining the trajectory of the hands. The six control parameters are mapped to animation parameters that modify the timing and position of the control points in the spline or set the tension, bias and continuity of the spline. Their control parameters are intuitive, but not independent, specifically they mention an unresolved conceptual interdependence between the power and temporal extent (roughly duration) control parameters.

2.3.1.2 Parameterization using Constraint Editing

Recorded motion primitives can be modified to adhere to a pose constraint, using a motion modification technique (see Section 2.2.1.1). In this case, the animation parameters are used directly as control parameters. Le Callennec et al. [44] provide a PFIK+F framework that can handle multiple pose constraints. It resolves possible conflicts in constraints by satisfying those with the highest priority first.

Amaya et al. [8] state that emotion is observed in motion timing and spatial amplitude. An emotion transform is applied on neutral motion using non-linear time-warping and a spatial amplitude transform technique based on signal amplifying methods. The required time warp and amplification for such an emotion transform is obtained by determining the emotional transforms needed to get from recorded neutral movement to the same movement executed in an emotional style. Hsu et al. [117] describe a similar method for emotion transform, using a Linear Time Invariant model rather than signal amplification for the spatial transform.

2.3.1.3 Parameterization using Blending

To achieve a desired pose at a desired time, a set of motion primitives to interpolate and their interpolation weights have to be found. Many parameterization methods have been developed to solve a subset of the pose constraint problem: positioning an end effector at a desired position $\mathbf{s}_{des}$, specified by three control parameters. Unfortunately, blending does not yield a linear parameterization of this control parameter space [242]. That is, if $\mathbf{s}_{des}$ is exactly in between $\mathbf{s}_1$ and $\mathbf{s}_2$, this does not mean that a blend with interpolation weights of 0.5 of the joint rotation vectors $\mathbf{q}_1$ and $\mathbf{q}_2$, placing the end effector at $\mathbf{s}_1$ and $\mathbf{s}_2$, will end up placing the end effector at $\mathbf{s}_{des}$. Several methods have been developed to solve this discrepancy.

Rose et al. [240] use a scattered data interpolation method to approximate control parameter values. This method calculates the resulting motion primitive using a linear map between blend weights and control parameters combined with radial basis functions centered on each example motion primitive. For desired poses

that are far from the examples, the motion primitives calculated using this approach are based purely at the linear approximation and hence are effectively arbitrary [157]. Grassia [96] uses a linear approximation of the blend weights in an initial positioning step and then exactly positions the end effector at the goal position using a constraint based method (see Section 2.2.1.1). Van Basten et al. [20] linearize part of the posture representation and interpolate *positions* of joints instead of *rotations*. This will result in end effectors that are exactly on the desired position.

Many other techniques make use of pseudo example motion primitives. Wiley and Hahn [305] *resample* the examples to a dense regular grid in a precomputing step that exhaustively searches through interpolation weights and recorded motion primitives. The grid can then be used to efficiently select the pseudo examples to be interpolated. Note that, compared to Rose et al. [240], only a subset of the example motion primitives are blended. Also, the number of required example motion primitives is $O(2^d)$, where d is the dimensionality of the parameter space, whereas Rose et al. [240] only require $O(d)$ samples. Rose et al. [242] use the smoothness of the function that maps blend weights to end effector position values to create pseudo examples online at selected positions, iteratively improving the accuracy of the parameterization. Kovar and Gleicher [157] randomly create random pseudo examples online. By using k-nearest neighbor interpolation rather than interpolating from all samples, the run-time cost of their algorithm is independent of the number of recorded and pseudo example motion primitives.

Using blending methods, the intensity of an emotion or physical state can be adapted. For example: by blending a happy walk with a normal walk, a slightly happy walk can be obtained [120, 240]. Unuma et al. [296] introduces blending in the Fourier domain for cyclic motions (such as walking and running). Such a Fourier domain blend ensures that the motions that are to be blended are automatically time-aligned, so time warping is not needed in the preprocessing steps. For walking and running, the Fourier description provides parameters to control the step size, speed, duration of the flight stage and maximum height during the flight stage. Similar motions with different emotional or physiological aspects (brisk, tired, happy, etc) can be blended in the Fourier domain, so that these aspects can be used as motion parameters. Fourier descriptions can also be used to transfer motion aspects: by applying the Fourier description of briskness from a brisk walk onto a normal run, a brisk run is created. Because the parameters are qualitative, strict accuracy cannot be attained by the blending methods described above.

Torresani et al. [288] provide parameterization of three of the LMA Effort control parameters (see Section 2.3.1.1). Recorded motion primitives are annotated by an LMA expert. These annotations are translated into numerical values. By annotating the Effort of blends of motion primitives with known Effort values, a function that maps blend weights, input joint angle data and input Effort values to the output Effort values is learned. A motion primitive with unknown Effort values can then be adapted to have desired Effort values by blending. This entails finding its k-nearest neighbors in the database of annotated motion primitives and finding the motion primitive pair that, with the optimal blend weight (found by uniform sampling), approximates the desired Effort values the best. At the cost of computation time

and annotation effort (by an LMA-expert), this method achieves a more accurate Effort parameterization than simple linear blending.

2.3.1.4 Parameterization in Statistical Models

Satisfying control parameters using statistical models requires specialized methods for each statistical model. Grochow et al. [100] search their SGPLMVM model representation of the motion space using optimization to create motion primitives that satisfy pose constraints. Li et al.'s [173] motion texton representation of the motion space allows the creation of motion primitives by directly specifying poses at selected frames. Mukai and Kuriyama [199], create a geostatistical model of a set of recorded motion primitives. Geostatistical interpolation is used to create the motion primitive with desired pose constraints. This method is more accurate in achieving the desired pose constraints than blending methods that use radial basis functions. It is more efficient (in terms of calculation time and memory usage) than blending methods that use pseudo examples. Carvalho et al. [46] introduce a constraint based editing method that uses the same prioritized IK solver as [44] (see also Section 2.3.1.2) on a low-dimensional statistical motion model, generated using PCA or probabilistic PCA. Their system is computationally more efficient, and is, according to the authors, in some cases more natural than the PFIK+F approach used in [44].

In human motion, there are many correlations between joint actions. Statistical methods [69] and machine learning [38] have been employed to find orthogonal control parameters in a set of recorded motion primitives. Because the parameters are independent, it is not necessary to resolve parameter conflicts. However, the control parameters learned in such approaches are not very intuitive to use and are highly dependent on the training data.

2.3.1.5 Parameterization using Physical Simulation

The desired state of a controller can be used directly as a control parameter. Animation parameters such as desired joint rotation, pelvis height or CoM position provide intuitive control. However, satisfying pose constraints precisely and timely using a physical controller is still an open problem, since in general it is unknown if and when a controller achieves such a pose constraint. Some recent efforts have attempted to address this issue. Neff et al. [207] use empirically determined offsets on the pose time and angular span multipliers on the pose itself, so that their system achieved poses on time, for certain classes of movement (e.g. gesture). Other systems rely on critically damped controllers to achieve precisely and timely arm poses [6, 149]. These controllers can only generate movement in which the 'muscles' are critically damped and impose limited or no movement of the trunk.

Abe and Popović [1] show how to set up a physical controller that satisfies control parameter values in order of their priority. They report that prioritization of balance control interferes with posture control, which makes it difficult to combine these two control objectives in a natural manner. In later work, Abe et al. [2] use a

weighted combination of control objectives to achieve a compromise between their control parameter values. The weights can be used to move smoothly from one objective to the next. Some techniques have been devised to map control parameters to animation parameters used in controllers.

Chao et al. [56] provide a mapping from LMA-Effort parameters to animation parameters for a PD-controller, such as damping, stiffness and desired joint rotation. Yin et al. [316] apply an optimized learning strategy to adapt the animation parameters $\mathbf{w}$ of a physical walking controller to a new situation parameterized by the control variable γ (for example: step over an obstacle of height γ, push a piece of furniture with weight γ, walk on slippery terrain with friction coefficient γ). The animation parameter space is searched for valid values of $\mathbf{w}$ (as in, those that do not make the virtual human fall) that achieve γ. There may be many viable solutions of $\mathbf{w}$ that achieve γ. A hand-authored objective function evaluates $\mathbf{w}$ to help select a unique optimal solution that achieves γ. It can be designed to prefer solutions that have a minimal deviation from the original animation parameters, a certain walking speed or step size, and so on. The learning process is off-line, but the learned animation parameter values can be interpolated to achieve real-time control. It is yet to be seen wether and how this method generalizes to more than one control parameter.

2.3.2 Concatenating

Concatenation (see Figure 2.4) deals with the generation of a sequences of motion primitives, to form a natural motion that satisfies certain control parameters and assures the naturalness and smoothness of the resulting motion. The motion primitives can be generated by different techniques (or the same technique that is initialized differently).

2.3.2.1 Concatenation using Motion Editing

Ease-in ease-out interpolation between two motion primitives can be used to concatenate them. The first motion primitive is faded out as the second one is faded in. Displacement maps (see Section 2.2.1.1) can also be used to transition from one motion primitive to another, as is done in [168]. Transitions between different pairs of motion primitives concatenated in this manner differ in naturalness. Ikemoto et al. [121] generate transitions by cached multi-way blends. They cluster recorded motion primitives using the distance metric by Kovar et al. [158]. All mediods (central item of cluster) are representatives for the motion primitives belonging to that cluster. During preprocessing, the naturalness of all possible 2, 3 or 4 multi-way blends between representatives is evaluated (using footskate and ZMP position as evaluation criteria) and the best blend recipe (containing a weight function and representatives) is stored. A transition is generated at runtime by matching the current and next motion primitives to mediods and applying the stored blend recipe.

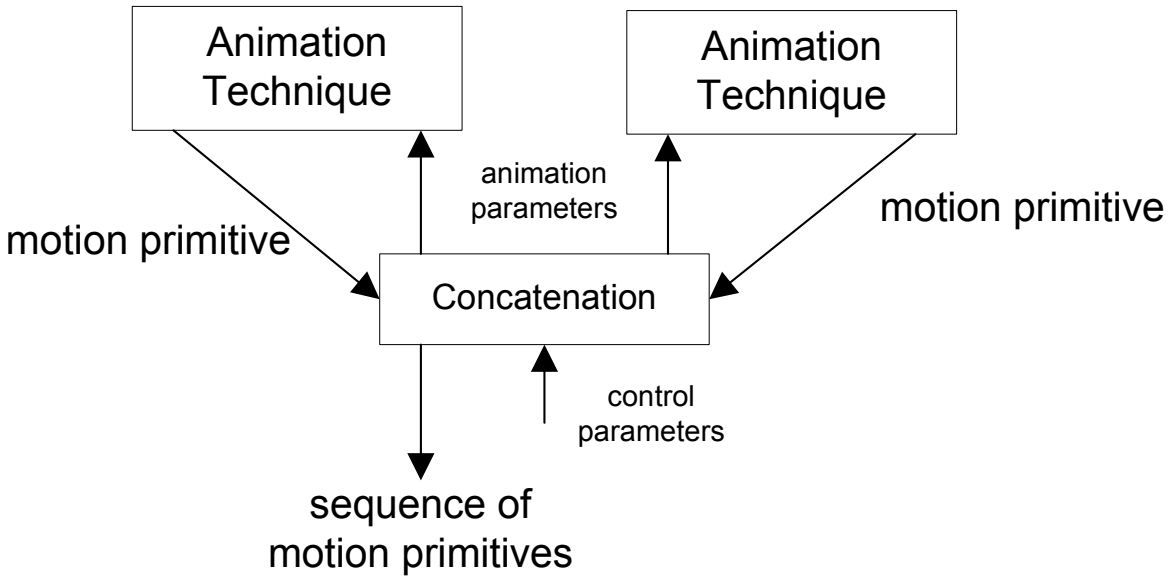


Figure 2.4: Concatenation generates a sequence of motion primitives using (possibly different) animation techniques. The resulting motion satisfies the control parameter values.

Motion Graphs In many applications, one requires multiple concatenated motion primitives to satisfy a longer term control parameter (for example: walk to a certain position). A very common technique is to encode the possible transitions between motion primitives in a graph-like structure: a motion graph. A motion graph is a directed graph where all edges correspond to motion primitives and nodes correspond to poses. Interpolations between poses from different (or the same) motion primitives that are 'similar enough' are added as new edges. In the game industry such graphs, move trees, were originally created manually [196]. Kovar et al. [158] present an algorithm that automatically creates motion graphs. Good transition points are automatically detected using Cartesian joint distance as a distance metric. After the graph is created, the graph can then be searched to find a sequence of motion primitives that adheres to control parameter values (for example: walking along a specified path). Many variations of motion graphs exist, which can be distinguished in off-line methods where the desired control parameters are known in advance and the motion is generated off-line (for example: [10, 158]), and methods that work at interactive speed (for example: [19, 93, 168, 169]). These techniques mainly differ in the graph structure or the search strategy. Here I mainly discuss the inherent naturalness-control-calculation time trade-offs in motion graphs. For an exhaustive literature overview I refer the reader to a recent article featuring motion graphs (e.g. [322]) or Forsyth et al.'s survey ([79], p184-194). Throughout this chapter, I make use of the terminology for naturalness, control and calculation time aspects of motion graphs introduced in [79].

At interactive speed, a global search on the graph is infeasible [79]. Local search evaluates only the values of control parameters in a path through a limited number of nodes when choosing the next sequence of motion primitives. Even if a path on the graph that satisfies control parameter values is available, a local search method

might not find it, because it cannot look far enough ahead. This is called the horizon problem in [79].

Reinforcement learning (first proposed by Lee and Lee [169]) can be used to learn (near) optimal long-term plans for specific control parameter *values* that are specified as a reward function. The learning process is off-line. For each global state (=world state $\times$ virtual human state) a (near) optimal path on the motion graph is learned that achieves specific selected control parameter values. Some flexibility can be gained by a smart selection of state and objective function. For example: if the state is set as the angle between the current walk direction and the goal direction, walking in any goal direction can be learned by learning how to walk forward. Walking to a desired 2D location can be learned in a similar manner. One can also learn a grid of control parameter values [169]. Because of its discretization of control parameter values, reinforcement learning sacrifices some accuracy for long-term goal satisfaction. Furthermore, it can be hard to coordinate multiple control parameter values and is typically very memory intensive [169]. Recent approaches using reinforcement learning aim to address the latter [172, 175, 292].

Control and motion planning is limited by the available paths on the graph. As more control parameters are added, less paths will become available that satisfy all their desired values. It is possible to extend the graph (and hence, gain more control) by adding more transitions. Unfortunately, at some point the added transitions become unnatural [18]. This is a typical trade-off when using motion graphs. More transitions will result in more control but also more visual artifacts such as footskate. Another disadvantage is that motion graphs are, in general, not able to generate motions that require tight coupling to the environment unless exactly those motions are in the database.

Several techniques have been developed that are able to automatically identify natural transitions between motion spaces. Shin and Oh [266] present fat graphs. In these fat graphs, blend spaces are constructed using edges that start and end at a common pose (or hub) of a motion graph. These blend spaces allow more flexible parameterization than traditional motion graphs. However, in order to transition from one motion to another, the virtual human must always first move through one of the common poses. Heck and Gleicher [107] introduce parametric motion graphs. A parametric motion graph encodes an edge as a mapping from a control parameter in the source motion space (creating a source motion primitive) to a subspace of the control parameter values in a target motion space. This subspace of parameter values in target space is selected so that they create a target motion primitives that connects it to the source motion primitive (that is, the start of the target primitive and the end of the source primitive are 'similar enough'). Transitions are selected by specifying the current motion space and control parameter values and the target motion space and desired target control parameters values. If a natural transition satisfying the target control parameters exists, they are achieved precisely. If not, a transition that provides the closest match to the target control parameter values can be selected. This either sacrifices accuracy for naturalness (for example, for a punch motion space with target punch position as a control parameter), or it can sacrifice responsiveness for naturalness if control parameter values are achie-

ved by subsequent transitions (for example for a walk motion space with a direction control parameter).

Concatenation using Statistical Methods Li et al. [173] model a motion space as a linear dynamic system (LDS). They define a distance metric for LDSs and construct a motion graph-like structure to support concatenation of similar LDSs. By setting the first two poses of the next LDS in the path to the last two poses of the current LDS (see Section 2.3.1.4), a fluent connection is achieved.

2.3.2.2 Concatenation of Physically Controlled Motion

In physical simulation using controllers, concatenation implies a switch to a different controller. If the exit state of one controller leaves the simulation in a valid entry state for the next controller, valid transitions can easily be attained [311]. Predefined transitions between controllers that satisfy this condition can be encoded in a state machine. For example, [115] shows a state machine that uses different phases (and thus, controllers) for the flight, loading, heel contact, heel and toe contact, toe contact and unloading phases of a running motion.

A transitional controller can be designed to facilitate transitions between controllers with incompatible exit and entry states. Wooten and Hodgins [311] demonstrates this, by using a landing controller to take a virtual human from a flight state to a state suitable for balancing on the ground. Faloutsos et al. [73] facilitates transitions between controllers by describing preconditions and post-conditions for each controller. The preconditions define the sensor values (see Section 2.1.3.1) that lead to a successful execution of the controller. Specifying valid preconditions for controllers is not always a trivial task (for example: what are valid preconditions for balancing?). Support vector machine (SVM) classifiers are trained to predict the success or failure of a controller given sensor values. The preconditions for a controller are then determined by what a trained SVM for that controller classifies as successful.

Coros et al. [61] show how to create control policies that satisfy longer term goals. The policy selects a near optimal sequence of locomotion controllers given a certain control parameter *value* offline. Each controller executes one locomotion cycle. After each walk cycle, the control policy selects the controller that will achieve a new global state (=world state $\times$ virtual human state) that maximizes the reward. Rewards are explored for 'trusted' global states (those close to states achieved 'normally' in the controllers), using off-line reinforcement learning, in a similar manner as discussed earlier for motion graphs (Section 2.3.2.1).

2.3.2.3 Concatenation of Procedural Motion

Zeltzer [319] models the different phases of a procedural walking motion by different procedural models and concatenates them using a state machine. Some frameworks for the generation of procedural arm gestures concatenate the gestures using procedural techniques that allow a flexible start pose of the arm [105, 155]. The

end pose of the previous gesture is then used as the start pose of the current gesture. Other procedural animation systems use interpolation to generate a transition motion primitive between two procedural motions [131, 218].

2.3.2.4 Concatenating Physical Simulation and Motion Editing

Motion editing techniques provide natural motion, but it is hard to set them up to interact with the physical world. Physical simulation provides world interaction, but less naturalness. Several methods have been developed to take advantage of the strength of both techniques by switching between them depending on the type of interaction needed.

Shapiro et al. [262] switch control from kinematics to physics on contact with physical objects in the environment. A transition from physical simulation to motion editing (in their system a motion graph) can be made if the pose of the virtual human is similar to a pose in a motion primitive of one of the motion editing motion spaces. It is not stated how such a suitable motion primitive is found. Presumably the number of motion primitives in the graph is low, so that an exhaustive search can be performed on all their poses. Mandel [187] makes the transition from motion editing to PD-control, whenever some physical event occurs that makes the virtual human fall over. A PD-control system is then started in the pose last set by the motion editing motion primitive. A fall controller lets the virtual human fall, while trying to break this fall with the hands. As soon as the hands hit the floor, the system attempts to return control to motion editing. To find a suitable motion primitive, the motion capture database is searched for a motion primitive that has a similar pose to the pose the virtual human is in. This is done using the Approximate Nearest Neighbor Search algorithm. An intermediate physical controller then moves the virtual human to this pose. Once the virtual human is close enough to that pose, control is returned to motion editing. NaturalMotion's [200] Euporia and Endorphin animation systems allow transitions between motion editing and physical simulation. Selecting a suitable motion primitive to play after physical simulation is left to the motion author.

Rather than using recovery controllers, Zordan et al. [326] search a motion capture database to find a suitable recovery motion primitive to play after a physical impact. During the physical impact, a physical rag doll motion is played for a short period of time (0.1-0.2s), then motion is steered by a physical tracking controller (see Section 2.4.1.3), that tracks a blend of the motion primitive before the impact and the selected motion primitive after the impact. In later work, Zordan et al. [325] contribute an automatic, real-time, motion primitive search algorithm. Reentry motion primitive candidates are classified off-line, using machine learning. This significantly reduces the number of candidates to select from.

2.3.3 Combination

A virtual human can execute multiple tasks that each require motion at the same time, possibly with different parts of the body. Rather than explicitly setting up new

motion primitives for each combination of motion acting on a separate body part, different motion primitives, possibly created by different animation techniques, can be combined in such a way that a coherent whole body motion results (see Figure 2.5).

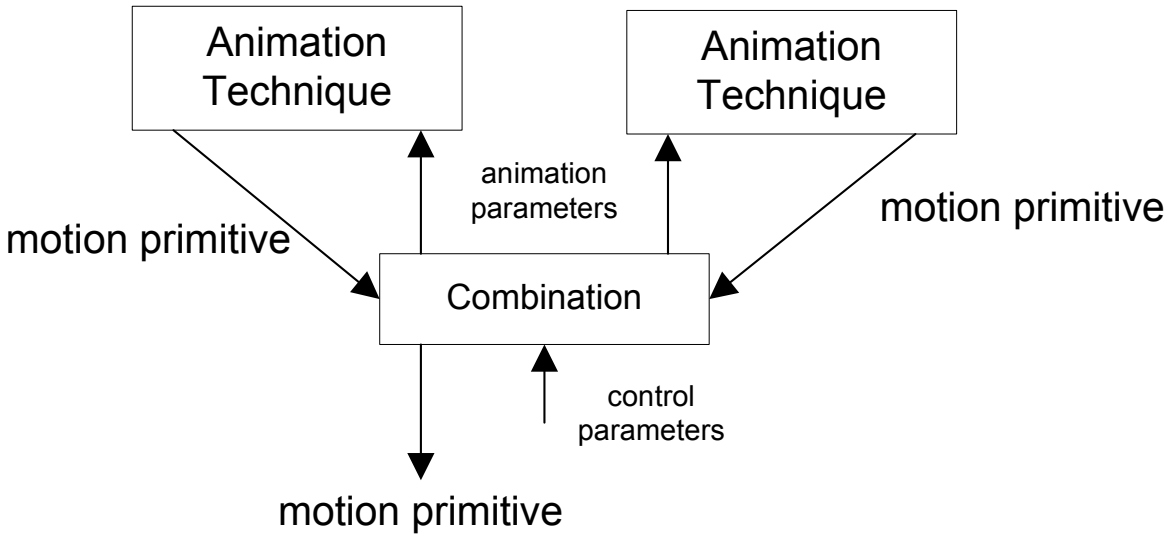


Figure 2.5: Different motion primitives, each acting on a subset of the joints of the virtual human can be combined to form full body motion.

2.3.3.1 Combination using Motion Editing

A simple way to combine motion primitives is by using a direct 'ease-in ease-out' interpolation of the motion primitives (as done in [131, 217]). The interpolation weights of the motion primitives to be combined is set per joint, so that certain motion primitives can be set up to affect certain joints more than others. This method can produce unrealistic results because it ignores both physical and stylistic correlations between the motion of various joints in the body [108].

Heck et al. [108], aim to combine (splice) one motion primitive on the upper body with one acting on the lower body. Both motion primitives contain a walk cycle. Temporal relations between the upper and the lower body are maintained by making use of the rhythmic nature of walking to time warp and align the motion primitives. The pelvis is rotated in such a way that the upper and lower body are aligned, while retaining the desired upper body posture. Ha and Han [101] generalize Heck's splicing method. They construct a time warp between upper body and lower body motion spaces off-line. This time warp can then be used to splice motion primitives of the two spaces online. Note that these two splicing methods only enforce coherence of the upper and lower body in the *temporal* domain.

2.3.3.2 Combination of Physical Controllers

Physical controllers can be combined by adding up the forces and torques applied by them on each joint (as done in [312]). Such a combination automatically provides

physically coherent whole body motion, because an articulated rigid body system models the force transference through the joints.

2.3.3.3 Combination of Procedural Motion

Many procedural animation systems combine procedural motion on different body parts, by employing a procedural motion technique for each body part [105, 155]. Procedural motion from different procedural models, acting on the same body part can be combined using interpolation (see Section 2.3.3.1, [218]). Thiebaux et al. [280] employ specialized blend mechanisms to combine motion primitives generated by different procedural animation techniques.

2.3.3.4 Combination of Kinematic Motion and Physical Simulation

The requirements of physical integrity and accuracy are often of different importance for different body parts. For example, for a gesturing virtual human, positional and timing accuracy is primarily important on movement of a gesturing arm or head. At the same time, a physically valid balancing motion of the whole body could be achieved by moving only the lower body, where precise timing is less important. Combining kinematic motion with physical simulation on different body parts allows one to combine the accuracy of motion generated by procedural animation or motion editing with the physical realism of physical simulation. Oore et al. [211] present a system that mixes physical simulation, acting on the knee and ankle joints, with kinematical upper body motion. The physical model is coupled with the upper body through its mass displacement. The joint torques of the kinematically moved parts in the upper body are not taken into account in the physical movement of the lower body. Isaacs and Cohen [123] show how inverse and forward dynamics (see Appendix A) can be combined in a custom designed physical simulation system, given that either the joint accelerations or the joint torques are known for each joint, at each frame. This way, if kinematic motion is known at every frame for some joints, the forces those joints exert on the other joints is taken into account when the remaining joints are moved using physical simulation. In Chapter 3, I present my extension on this work that provides a simplification of the simulation model. My system allows the use of efficient iterative techniques to calculate the torques exerted by the kinematically steered joints and provides easy integration with existing physics engines.

2.3.3.5 Combining Procedural Motion and Motion Editing

By augmenting motion editing with procedural motion, expressiveness can be enhanced without requiring a prohibitive amount of motion primitive examples. Some examples: a biomechanical model of eye movement (which is hard to motion capture) can be combined with a motion editing technique for neck and trunk movement [165]. Heck [109] employs a biologically and psychologically inspired

model for gaze that is layered on top of motion primitives created by motion editing. Shapiro et al. [261] combine lower-body motion capture with arm movement determined by planning techniques from robotics.

2.3.4 Aspects of Control

In the previous subsections I have looked at ways to parameterize, concatenate and combine motion spaces using various techniques. Here I discuss how much control can be gained using such techniques, by looking at the various aspects of control.

2.3.4.1 Responsiveness

Responsiveness determines how fast a desired change in the motion plan is achieved. Responsiveness is a major theme in the design of motion graphs, it might take a while to traverse the graph to reach the desired node, especially if the graph is sparse. Forsyth et al. [79] introduce the diameter: the average path length of the shortest path connecting two nodes on a motion graph as a measure for responsiveness. A denser graph (with a smaller diameter) can be created by sacrificing some naturalness (see Section 2.3.2.1). Physical simulation has high responsiveness to physical events (for example, being hit by a falling anvil), but lower responsiveness to control parameter changes that effect the desired state of the virtual human. Procedural animation and motion editing techniques have higher responsiveness to parameters that change the desired state of the virtual human, but direct reaction to physical events that occur in the world is not built-in.

2.3.4.2 Precision

Precision is the accuracy with which control parameters (such as end effector position or timing constraints) are achieved. Procedural motion is very precise. Motion editing techniques can provide precision at the cost of calculation time. Physical simulation is imprecise, it is unknown whether and when desired pose and time constraints have been met. Some precision can be gained by sacrificing naturalness and creating only motions in which the 'muscles' are critically damped (see Section 2.3.1.5).

2.3.4.3 Coverage

Coverage deals with how much of the control parameter space is covered. Motion graphs can suffer from bad coverage. For example: some parts in an environment cannot be reached from certain nodes in a motion graph because no path starting in this nodes will go there. Reitsma and Pollard [233] present an algorithm to determine the environment coverage of a given motion graph. Note that the coverage of a motion graph is also greatly influenced by the search algorithm it uses. Even if a path on the graph that satisfies control parameter values is available, a local search method might not find it. Physical simulation can suffer from bad coverage

whenever control parameter values create motion primitives that put the virtual human on the edge of balance. The coverage of physical simulation can be increased by sacrificing some balancing naturalness (see [313]) or by using better balance methods that require more calculation time (see [268]). While most motion editing techniques can cover a wide range of control parameter values, only the control parameters that result in a motion primitive near an example motion primitive will yield natural motion. Procedural motion has good coverage, but again not all control parameter values will provide natural motion.

2.3.4.4 Expressiveness

I have defined expressiveness as the number of control parameters that can be used in the motion plan. Procedural and physical simulation techniques have high expressiveness. The number of parameters that can effectively be used in motion editing is low.

2.3.4.5 Intuitiveness

Intuitiveness deals with how intuitive the control parameters that can be used in the specification of the motion plan are. All techniques allow the use of control parameters that can set pose constraints. Other control parameters (such as emotion, physical state) can often be mapped to animation parameters. An intuitive set of control parameters might cause conflicts between animation parameters, but an orthogonal set of control parameters is typically not intuitive (see Section 2.3.1.4). For example, [38] reports having a parameter that sets both the speed *and* the global pose. Therefore, orthogonal control parameters are typically used solely to create small variations on existing motion (see Section 2.4.4.2).

2.3.4.6 Control Enhancement with Multiple Animation Paradigms

By combining and concatenating motion primitives created by different animation techniques, several aspects of control can be enhanced. For example, a concatenation of a motion primitive created by motion editing with one created using physical simulation enhances the responsiveness to physical events (see 2.3.2.4). Another example is the enhancement of expressiveness by combining procedural motion and motion editing (see Section 2.3.3.5).

2.4 Naturalness

For many animation systems, plausibility or naturalness rather than full realism is acceptable. I define naturalness as perceived realism of a virtual human's movement. Naturalness therefore partly depends on properties of human observation.

Physical realism is one property of natural animation (see also the Appendix A), but physical realism alone is not enough for motion to be perceived as natural. In-

volvement of the whole body is crucial to make an animation natural [58]. Furthermore, movement should be consistent with static (such as age, gender) properties of the virtual human that is being animated [98, 246]. Variability is a concern if a motion is to be repeated. In this section I will elaborate on these different aspects of naturalness and show how naturalness can be enhanced and evaluated. Naturalness effects related to animation planning, such as the plausibility of the motion with respect to the cognitive and emotional state of the moving virtual human [98], are beyond the scope of this thesis.

2.4.1 Physical and Biological Realism

Motion primitives created by physical simulation techniques are physically realistic by design. It is relatively easy to consider muscle strength in these methods. Motion captured animation is also physically realistic, since it originates from real humans moving. However, motion editing might invalidate its physical correctness, introducing artifacts such as foot skate, unnatural balance, or momentum changes in flight. I outline some methods to correct or prevent these artifacts and enhance physical and biological realism.

2.4.1.1 Physical filters

The physical naturalness of motion primitives can be improved by post processing motion primitives with a physical filter.

For instance, Pollard and Reitsma [222] propose to filter motion primitives to obtain physically correct ground contact. A friction model is used to make the foot slide when appropriate. Their filter makes use of the fact that a (virtual) human cannot apply a force or torque at its root joint. Each frame of motion is cast on a physical model of the virtual human. Then, per frame, the net root forces and torques are eliminated by modifying the rotational acceleration on all actuated joints and the rotational and transitional acceleration on the root.

Shin et al. [265] employ a constraint based motion editing method (see Section 2.2.1.1) to enhance the physical and biomechanical correctness of edited motion. During flight stages, the angular momentum is conserved and the center of mass is constrained to follow a parabolic path. During ground contact, the ZMP is constrained to fall into the support polygon. The corrections are applied to a user-selected set of joints during the flight stage, ZMP correction is applied on one user selected joint.

Footskate is a typical artifact caused by motion editing. The virtual human's foot slides on the floor after the virtual human plants it, rather than remaining tightly in place [122]. If it is known when a foot is planted, then a constraint based motion editing method (see Section 2.2.1.1) can be used as a motion filter, to constrain the movement of the planted foot [159]. Fully automatic reliable detection of footskate in real time is still an open problem. Existing methods have to be trained for each motion [122] or refine motion type (e.g. run, walk) specific estimations of contact times and durations [89]. Alternatively, foot contact can be annotated in the

recorded motion primitives, and motion editing techniques can be set up to retain these annotations [156].

2.4.1.2 Retaining Physical Correctness in Interpolation

Because of the difficulties and large computation time associated with physical filters, some interpolation techniques deal with physical realism during the interpolation stage, rather than using a post-processing method. A number of simple modifications can be used to improve the physical correctness of interpolation of motion primitives that are physically correct on their own [250]. By interpolating the center of mass, rather than the root and clever selection of the interpolation duration, the net force during flight is equal to gravity. If, during ground contact, the center of mass, the foot positions, knee-swivel angles and all joints angles except the legs are interpolated, rather than directly interpolating joint rotations, the feet will not penetrate the ground, balance will be retained and the ground friction will be within the same friction cones as the source motion primitives.

Ménardais et al. [192] use a simple technique to avoid or reduce footskate. Motion primitives are annotated with support phase information (left foot, right foot, double support, no support). A time warp then synchronizes the support phases of motion primitives so that they are compatible during the interpolation. Treuille et al. [292] prevent footskate in support phases where only one foot is on the ground by first aligning the support foot of the second motion with the support foot of the first and then interpolating the motion primitives with the support feet as the root. Oshita [212] contributes a method to generate transitions between two motion primitives based on their support phase that does not require re-aligning them and can handle a wider range of support phase combinations. It uses Treuille et al.'s method for the connection of motion primitives in which the same foot is moved. Flying motions are connected by aligning their pelvis directions and interpolating some frames of the start of the second motion with some frames of the end of the first. A transition from a motion primitive with single support to one with double support is created by interpolating from some frames of the end of the first motion primitive with the start *pose* of the second motion primitive. Transition between double support motion primitives are created by modifying the lower body of the second motion primitive, so that its feet positions match the first motion primitive. The upper bodies are then interpolated. As soon as a foot is lifted in the second motion primitive, the lower body is interpolated with the second motion primitive too.

2.4.1.3 Improving Physical Correctness using Tracking

Tracking is used to enhance existing kinematic motion to allow physical interaction with the environment. A physical tracking controller tracks the joint rotation path specified in a motion primitive. This is done by setting this path as the desired state for the controller. Physical tracking recently became a component of some commercial high level animation toolkits [11, 106].

A tracking PD-controller necessarily has very high PD-gains, which result in stiff reactions to the environment. The PD-gains can be reduced on impact, to decrease such undesired stiffness (as done in [313, 324]). More sophisticated controllers use a predictive model that determines joint torques and typically corrects small perturbations using a low gain feedback PD-controller (for example: [198, 269, 315, 317]).

Motion capture noise, retargetting errors, tracking errors and environmental changes can easily disturb the balance of a virtual human that is controlled by tracking. For early tracking methods such as [149, 323] this was not an issue because they only track the upper body. Other tracking methods enforce balance by constraining the root to the translation specified in the motion primitive [200, 315]. Zordan and Hodgins [324] use a balance controller specialized for standing with double support contact. Wrotek et al. [313] use a less realistic balancing method that does allow locomotion: a weak root spring connects the root of the virtual human to the world. This spring can 'break' if too much force is exerted on it, causing the virtual human to lose balance. Yin et al. [317] use a custom balance controller for locomotion. Da Silva et al. [268] use a linear time varying system that learns (in an off line process) a balance strategy from reference motions, which allows them to track both cyclic and non-cyclic motions. Muico et al. [198] do not make use of a balance controller, but make use of a more precise torque prediction model instead. Their non-linear predictive model takes contact forces into account and tracks the input motion precisely. It allows the creation of controllers for agile motions, including running and sharply turning.

Using off line learning from a given motion primitive to construct a balance strategy (as in [268]) or a forward model (as in [198, 317]) enhances the naturalness of the resulting motion generated by a controller. However, by using such off line strategies some control is lost, since they no longer allow the tracking of unknown (for instance: generated by a motion editing technique) motion primitives.

2.4.1.4 Physical Correctness in Procedural Techniques

Physical simulation can greatly enhance expressive procedural motion. It can help to model important nuances of virtual human motion, such as realistic balance, force transference between limbs and momentum effects such as overshoot [201]. Physical controllers can explicitly address muscle strength and comfort. Some of these effects have, to some extent, been reproduced by procedural models.

Inverse kinetics [37] is a kinematic technique that can be used to position the CoM of a virtual human. This does help in creating balanced poses, but to generated realistically balanced *movement*, these methods need to be augmented with a model that provides a path of the CoM over time. Neff and Fiume [204] devise a feedback-based procedural balance system based on the physical controller of [312]. Unlike this physical balance controller, the procedural system works only on a single supporting foot and takes just the position of the CoM and velocity of the CoM, but not the forces generated by upper body movement into account.

Inverse dynamics can be used to analyze the muscle power used in procedural

motion. The motion can then be adapted to adhere to muscle strength limits (as done in [145, 171]).

2.4.2 Whole Body Involvement

Procedural gesture animation techniques typically steer the head and the arms and leave the rest of the body relatively stiff. Naturalness can be enhanced by providing automatic, coherent movement of the rest of the body. Some of the techniques used to enhance physical realism also help to engage the whole body. For example, a physically based balance model can be used to automatically generate lower body movement (see Section 2.4.1.4 and [201]). In Chapter 3 I show how to combine procedural animation with physically controlled balancing to achieve involvement of the lower body.

Egges and Magnenat-Thalmann [68] propose a statistical model to enhance the naturalness of procedurally generated gesture movement on the arms. PCA is performed on a motion capture (mocap) database of gesture animation. Using this PCA analysis, the procedural animation is filtered in PCA-space, in such a way that only movement similar to that in the database (and thus assumed natural) remains. Because the PCA components involve multiple joints, this automatically engages the full body. This method sacrifices some control —exact limb positioning is no longer guaranteed— for a more natural full body motion.

Both Chi et al. [58] and Neff et al. [206] aim to involve the torso automatically in gesture movement. The Effort and Shape parameters used to enhance the expressiveness of procedural gesture in [58] (see Section 2.3.1.1) are also used to enhance their procedurally generated torso movement. Neff et al. [206] show that 'drives', such as hand position and gaze direction can be used to automatically generate torso movement. This is done by defining a mapping between the drives and movement parameters of a procedural torso movement model.

2.4.3 Style

Style denotes the particular way in which a motion is performed. Stylistic differences of motion with the same function are caused by certain more or less static personal characteristics of the subject, such as age, gender and personality [98, 246]. It is important to endow virtual humans with style. Style contributes to naturalness, and, even more importantly, expresses information about the virtual human such as cultural identity, as well as his relationship to other (virtual) humans, such as role and power relationship. Style is reflected by the motions a virtual human performs and the manner in which these motions are performed [207, 209]. In this chapter I focus solely on the latter. I discuss techniques that achieve a certain style in real time. This can be done by post-processing generated motion primitives or by finding the control or animation parameter values that create a motion primitive in the desired style through an animation technique.

2.4.3.1 Style using Motion Editing

Motion capture also automatically captures the style of the motion captured actor. Ideally, this style could be isolated and be used to replace or define the style of other recorded or generated motions primitives. Here I focus on techniques that aim to do this automatically (in contrast to methods that require the animator to select the style component to transfer, e.g. [259]) and in real time.

Urtasun et al. [297] employ blending from recorded motion primitives from different subjects (and thus with different styles) in PCA space for style transition. A motion capture database is constructed, containing recorded motion primitives of several subjects, with different values for one control parameter (for example: jumping with different heights). A motion in the style of a new subject is created from a single recorded motion primitive of this subject. First, the recorded motion primitive is modeled as a blend of motion primitives from the different subjects in the database that have the same parameter value. The weights of this blend are then used to construct motion primitives with a new parameter values using a blend of motions in the database with these new parameter values. This system can create motion in the style of a user in an online application, by tracking the users movement using a cheap computer vision system.

Egges et al. [69] generate different styles of idle motion using recorded motion primitives of different individuals. On top of the posture shift motions, variation of movement is generated by applying a noise function on principal components derived from recorded motion primitives. This noise function is defined by a probabilistic model of recorded variations in motion. Individualized variations can be synthesized by determining the parameters of the probabilistic model for a given individual.

2.4.3.2 Style in Physical/Procedural Simulation

Procedural animation applies style by mapping style characteristics to lower level animation parameters, using parameterization. Perlin [217] models personality and emotion using noise functions on top of motion generated by an existing procedural model. Ruttkay and Pelachaud [246] model style as a mapping from static characteristics, such as age or sex to gesture animation parameters in a procedural animation system. Neff and Fiume [203] model style using a *Character Sketch*. Such a sketch defines modifications to be made to control parameters, can be designed to automatically insert new actions to an animation script and provides a default stance.

2.4.4 Variability

Variability is a measure of the differences in a motion which is repeated many times by the same person [33]. If the same motion primitive occurs several times in a motion performance, the performance will look unnatural. Several methods can be used to avoid this invalidation of naturalness.

2.4.4.1 Procedural Generation of Variability

Perlin [217] simulates variability by adding noise to the rotation of some of the joints in the skeleton of a virtual human. This method is not scalable on all joints, because relations exist between rotation of one joint and rotation of another. If these relations are not captured, the resulting animation will look unrealistic [69]. Bodenheimer et al. [33] apply variability by using a biomechanically inspired method. Since the amount of variability is usually correlated to the amplitude of the movements of the body (see Section 2.1.3.3), the noise has its largest amplitude at the extrema of a DoF of a moving joint. The noise is scaled with the distance the joint travels, thus obeying Fitts' Law. Since the shape of the noise is based upon the movement of the joints, this approach somewhat implicitly models inter-joint variability relations. However, reciprocally covarying movement variability between joints (for example an elbow movement to compensate shoulder variability on an aiming task) is not captured by this approach.

2.4.4.2 Generating Variability using Statistical Models

Statistical methods that capture orthogonal components of motion (such as [38, 69]) also capture the relation between joint movements. Since these components are independent, they can be modified separately. Small posture variations are generated by adjusting the components using Perlin noise [217]. In Li et al.'s [173] LDS model, variability is generated by sampling noise. Lau et al. [166] learn a motion space for the specific purpose of generating spatial and temporal variations of similar motion primitives, using a Dynamic Bayesian Network.

2.4.4.3 Generating Variability in Physical Simulation

Motion generated by physical simulation often looks 'sterile', because variation caused by small details is not taken into account [17]. Such details, for example small bumps on a floor, or the non-rigidness of human body parts are not simulated because it would not be possible to do so in real time or because simulation methods for this are yet unknown. Barzel et al. [17] propose some techniques to model some of these details in physically plausible (but not physically realistic) ways. For example: the inherent variability and instability of a physical simulation system can be exploited to generate motion variability by slight variations in its starting state, or a physical form of bump mapping can be used to create slight variations in the normal of a physically modeled flat floor.

Another, biological cause of variability in human movement is noise in the control signals that steer our limbs [103]. The variability of the noise increases with the torque to be exerted. Bodenheimer et al. [33] model this type of variability by adding noise to joint torques in a physical simulation in a similar way as described above for kinematic motion.

2.4.5 Evaluation of Naturalness

Measuring naturalness is a daunting task. It depends both on the properties of motion and on properties of human observation. Often, some control can be sacrificed to gain more naturalness, or some naturalness can be sacrificed to gain some needed control. I discuss how this naturalness-control trade-off can be quantified, provide some motion invariants and metrics that can be used to measure certain aspects of naturalness and discuss the setup of user tests that can measure naturalness as a whole.

2.4.5.1 Exploring the Naturalness-Coverage Trade-off

The naturalness of motion primitives created from the same motion space can vary with the control parameters that were used to create them. The relation between the size of the parameter space (coverage, see Section 2.3.4.3) and naturalness can be explored by having subjects directly set and evaluate parameter values, as done in [33]. Such an evaluation provides direct insight into the naturalness cost of a certain parameterization, or the control lost (specifically: reduction of coverage) if a certain level of naturalness is enforced. Clearly, having the subject determine the natural control parameter set is only feasible with a limited set of parameters.

2.4.5.2 Comparing with Motion Invariants

Some comparisons have been made by comparing motion invariants (see Section 2.1.3.3) of recorded motion with those of generated motion. End effector speed, end effector square jerk, end effector position and motion curvature can be used to compare human motion to generated motion, to evaluate how well invariants such as the bell shaped velocity profile, minimum jerk, Fitts' law and the two-third power law are modeled in the generated motion. So far such comparisons have been solely qualitative and were applied only in arm gesture domains; graphs of invariants in recorded motion were put side by side with graphs of generated motion (see [87, 155]).

2.4.5.3 Automatic Evaluation of Naturalness

Intuitively, physical correctness can be measured directly from the animation. Reitsma and Pollard [232] evaluate physical correctness by checking and evaluating perceptual metrics for allowable errors in horizontal and vertical velocities and the effective gravity constant for ballistic movement.

Metrics such as the average amount of footskate [4] and the number of frames in which the ZMP is outside the support polygon [124] address the physical anomalies in motion editing and can be used to compare the naturalness of different motion editing techniques.

Some attempts have been made to evaluate naturalness automatically. Ren et al. [234] argue that evaluation of the naturalness of human motion is not intrinsically subjective, but instead, an objective measure is imposed by the data as a whole. In

other words, movements that we have often seen are judged as natural, and movements that occur rarely are not. They make use of machine learning techniques, trained with statistical properties of human motion to classify new animations as natural or unnatural, and to point out the parts that invalidate natural movement. The system is still outperformed by human observers in recognizing natural or unnatural movement.

2.4.5.4 User Evaluation

One can (semi-)automatically evaluate certain naturalness properties of motion using automatic testing or motion invariant checking. However, most evaluation metrics check for a single naturalness artifact that only occurs within a specific animation technique. They can therefore not be used to compare different techniques. For example: it does not make much sense to evaluate the naturalness of physically simulated motion using a foot gliding metric, or to measure the naturalness of a procedural model that is specifically designed keeping a certain motion invariant in mind for adherence to that same motion invariant. Most naturalness metrics do not take human observation properties into account. User evaluation is invaluable for measuring naturalness as a whole and for providing between-technique naturalness comparisons.

Virtual humans do not usually have a photo-realistic embodiment. Therefore, if the naturalness of animation of a virtual human is evaluated by directly comparing moving humans with a moving virtual human, the embodiment could bias the judgment. A motion captured human movement can be projected onto the same embodiment as the virtual human. This projection is then compared with generated animation. Typically this is done in an informal way. A motion Turing Test is used to do this more formally (see [18, 55, 70, 115, 125]).

In a motion Turing test, subjects are shown generated movement and similar motion captured movement, displayed on the same virtual human. Then they are asked to judge whether this was a 'machine' moving or a real human. Methods from Signal Detection Theory [183] provide a bias-independent sensitivity metric d' that can be compared between different test setups, observers and motions. This metric indicates how well two motions can be discriminated. The d' found by comparing motion captured motion with the generated motion is used as a naturalness measure for the model based motion. This approach is followed in [55, 231, 232]. I refer the interested reader to [125] for an overview of test paradigms for the evaluation of naturalness of animation that can be used with Signal Detection Theory and their advantages and disadvantages.

Even if a certain movement is judged as natural, an invalidation of naturalness that is not noticed consciously can still have a social impact [227]. Unnaturally moving virtual humans can be evaluated as less interesting, less pleasant, less influential, more agitated and less successful in their delivery. So, while a virtual human Turing test is a good first measure of naturalness (at least it looked human-like), further evaluation should determine if certain intended aspects of the motion are being delivered. Such aspects could include showing emotion, enhancement of the

clearness of a spoken message using gesture, showing personality, and so on.

2.5 Discussion

I have discussed a variety of techniques that can all contribute to an 'ultimate' fully-controllable animation system producing natural motions in real time. Current techniques offer trade-offs between control, naturalness and calculation time. The selected trade-off depends on the application domain. Motion editing techniques employ the detail of captured motion or the talent of skilled animators, but they allow little deviation from the captured examples and can lack physical realism. Procedural motion offers detailed and precise control using a large number of parameters, but lacks naturalness. Physical simulation provides integration with the physical environment and physical realism. However, physical realism alone is not enough for naturalness and physical simulation offers poor precision in both timing and limb placement.

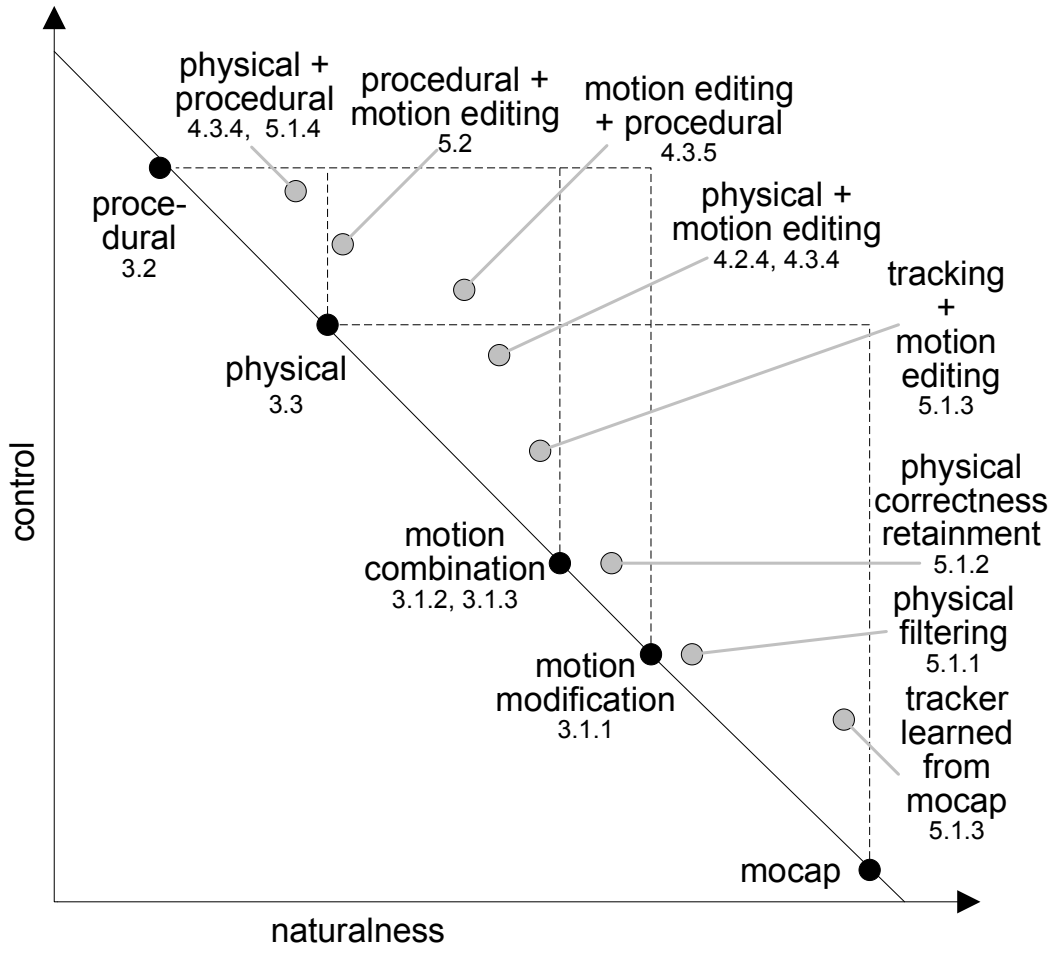


Figure 2.6: Control and naturalness of methods used in this chapter. Black dots indicate the animation techniques discussed in Section 2.2. Grey dots indicate hybrid methods.

A big challenge in the animation domain is finding an integrated way of gen-

erating natural motions that interact with the environment and provide detailed control. I have shown that hybrid systems that combine and concatenate motion generated by different paradigms can enhance both naturalness and control. These systems could provide a starting point for such an integration. In Figure 2.6, I provide a qualitative indication of the control and naturalness of the different hybrid systems discussed in this chapter. Note that the control provided by a hybrid system is typically not better than the best control of the two techniques it combines.[1] Similarly, the naturalness of a hybrid system is typically not greater than the naturalness of its most natural technique. The intersection of two dotted lines starting in an animation technique in Figure 2.6 indicates this best control and naturalness. Theoretically, very good naturalness and control could be achieved by combining techniques with high naturalness with those with great control. However, since techniques with great control also have low naturalness, it is hard to combine such techniques in a consistent manner.

I have shown different methods that execute animation plans generated by some higher level planning process. Such plans could be constructed by higher-level behavior generation mechanisms.

One domain of applicability of a flexible motion generation system is crowd simulation. Here physical characteristics of the environment (obstacles, quality of the ground) as well as physical and social behavior rules (e.g. strategy to avoid collision with objects and other people) serve as a basis for generating the animation plan.

This thesis focuses on another application domain: that of interactive virtual humans (see Chapter 10 for example applications). In this domain, the animation plan is typically constructed from intentions (such as greet the partner, indicate a location) and states (emotional, physical, cognitive). Typically the animation plan is embedded in a multimodal behavior script, describing the synchronization between speech and gesture. Recent efforts aim to unify the multimodal behavior scripts designed by different research groups into the Behavior Markup Language [152], discussed further in Chapter 6. In Chapter 3 I discuss the naturalness and control requirements for interactive virtual humans and introduce a hybrid method to combine the control of procedural (gesture) animation with the naturalness of physical simulation. Chapter 8 discusses Elckerlyc, an architecture that can schedule and execute an multimodal plan specified in BML and can make us of and combine the animation paradigms discussed in this chapter.

[1] In theory this is possible if the two techniques that are good in non-overlapping control aspects.

Chapter 3

Mixing Physical Simulation and Kinematic Motion[†]

Synthesis of expressive motion, such as conducting or gesturing for virtual humans in interactive, real-time applications is a challenging task. Such motion is often tightly synchronized with other internal output modalities, such as speech, or external output modalities such as user input or music. *Motion (capture) editing*[1] methods are not flexible enough to deal with the many control parameters and the tight synchronization with other modalities, that is needed for such expressive motion (see [92, 278] and Chapter 2). *Physically simulated* animation steers the body of a virtual human using muscle forces, taking gravity and inertia into account. While such motion is physically realistic, precise *timing* and limb positioning is still an open problem in real-time physical simulation (see Chapter 2.3.1.5). Therefore, timed expressive motion, as used in talking and gesturing virtual humans, is typically the domain of *procedural* motion techniques [58, 104, 207, 217, 304].

However, such procedural animation does not explicitly model physical integrity. As a result, the generated motion can look unnatural, as it does not seem to respond to gravity or inertia [185]. I refer the interested reader to Chapter 2 for background information on the animation techniques used throughout this chapter and a more elaborate discussion on the trade-offs between naturalness, control and calculation time of different animation techniques.

My system builds on the notion that the requirements of physical integrity and tight synchronization are often of different importance for different body parts. For example, for a gesturing virtual human, tight synchronization with speech is primarily important on the arm and head movement. At the same time, a physically valid balancing motion of the whole body could be achieved by moving only the lower body, where precise timing is less important.

My *mixed dynamics* system can apply kinematic motion (including procedural motion) on certain selected body parts, and combine this with physical simulation

[†]This chapter is largely based upon the article:
H. van Welbergen, J. Zwiers and Zs.M. Ruttkay. Real-Time Animation Using a Mix of Physical Simulation and Kinematics, Journal of Graphics, GPU and Game Tools, 14(4):1-21, 2009

[1]Or editing of keyframe animation

on the remaining body parts. Isaacs and Cohen [123] show how a combination of inverse and forward dynamics can be used to animate an articulated body in a physically coherent manner, if either the joint acceleration or the joint torque is known for each joint in the body, at each time frame. A similar approach is commonly used in biomechanics to visualize the biomechanical movement model of interest on some body parts (using joint torques), enhanced with known motion on other body parts (using kinematic motion) [213]. My system builds upon the ideas in [123] in an interactive application, using physical motion controllers and procedural kinematic motion. Implementing the system described in [123] entails implementing a full physical simulator. Implementing such a new physical simulator from the ground up is a daunting task. Nowadays, many physical simulators are available that handle the movement of articulated bodies, collision detection and friction (among others: [62, 106, 210, 264, 273]). I introduce a simplification in Isaac's simulation model, which allows the use of efficient iterative techniques to calculate the torques exerted by the kinematically steered joints. Using this simplification, my system can be used as plug-in for existing physical simulators.

In this chapter, the mixed dynamics technique is demonstrated on a virtual human by combining a physical controller for lower body balancing with kinematic animation for the upper body movements. I show how my algorithm is used with different types of kinematic arm and head animations, including parameterized procedural animation (for example, conducting motions or speech-accompanying gestures) and motion captured animation. This chapter will discuss the implementation of my system in detail, providing enough information for a robust implementation. Chapter 8.7 shows how this system is implemented in Elckerlyc.

Throughout this chapter I make use of Featherstone's concise notation of the equations of motion using 'spatial' 6-vectors. The transformation from such spatial vectors to the traditional 3-vectors is shown in Appendix B. For a more thorough overview of spatial algebra, I refer to [75].

3.1 Mixed Dynamics

In a mixed dynamics system (see Figure 3.1), motion is executed by a kinematic model (which can consist of motion editing method(s) and/or procedural motion model(s)) and by physical controller(s). The motion can be adapted in real time by changing the parameters and timing of the kinematic motion or the desired state of the physical controller. The kinematic model directly rotates the joints in the virtual human. I use inverse dynamics to calculate the torque applied by the kinematically steered body parts onto the physically steered body part, based on their rotations, angular velocities and angular accelerations. The physical controller calculates the joint torques that reduce the discrepancy between a desired physical state and the current physical state. An existing physical simulation engine is then used to calculate the joint rotations on the physically steered joints.

The body of the virtual human is divided in one physically steered part and one or more kinematically steered parts. Each part consists of joints, connected by rigid

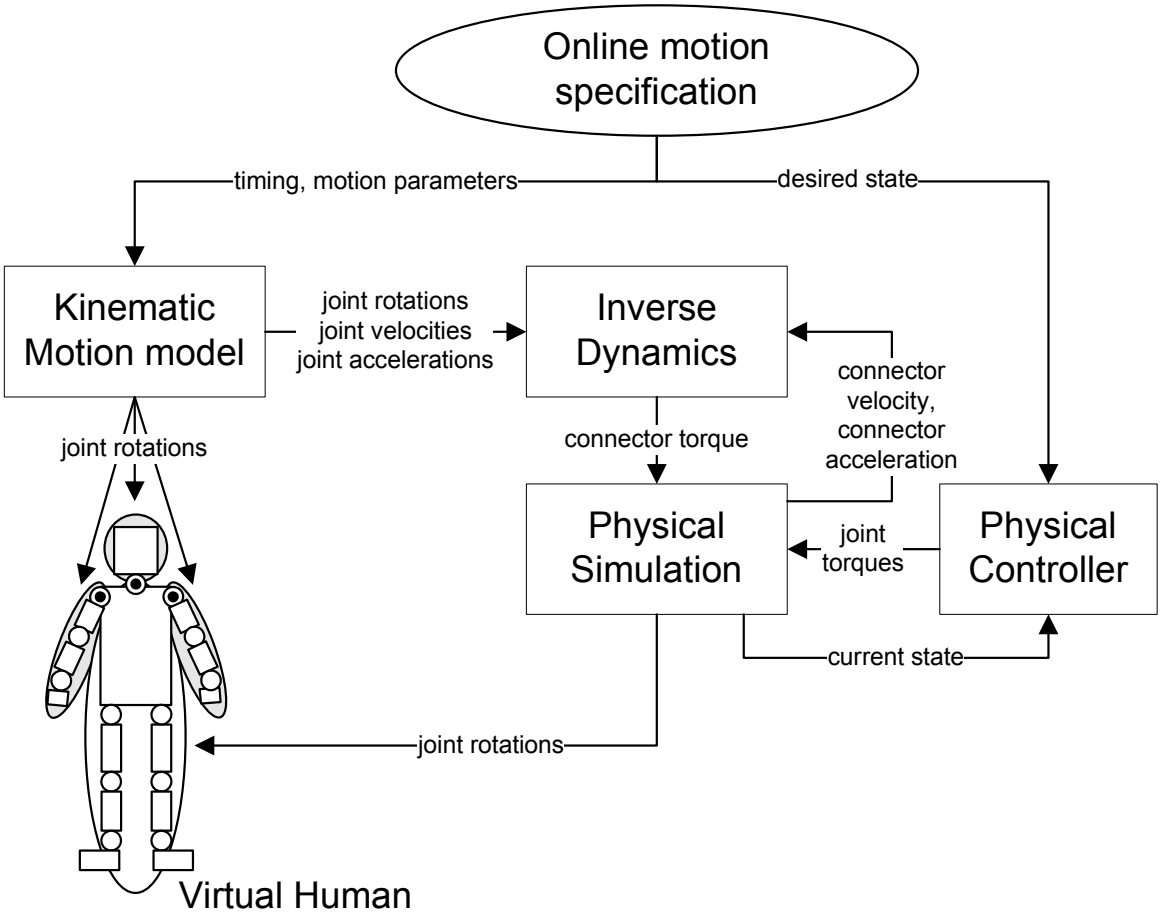


Figure 3.1: Mixed dynamics system.

bodies. I denote the set of joints on the physically steered part by P. In my example, these joints are located on the lower body. Groups of kinematically steered joints are denoted by $K_1, ..., K_n$. The joints in each K_j need to be connected to each other in a tree. P and all K_j's are mutually disjoint, that is, if a joint is steered, it is either steered by the kinematic model or by a physical controller. The groups are set up in such a way that each K_j connects to P at a single connector location C_j. C_j is located on the position of the root joint of K_j, in the rigid body in P that connects to this joint. See Figure 3.2 for an example structure.

To realistically model the effect that kinematic motion has on the physically steered body, the force exerted by each K_j is transfered to P via C_j. This force is calculated using inverse dynamics. The inverse dynamics algorithm needs the position, velocity and acceleration of each joint in K_j and the velocity and acceleration of C_j.

The velocity and acceleration of C_j is dependent on the movement of *all* joints in the body, and can only be calculated accurately by an algorithm that takes the accelerations $\ddot{\mathbf{q}}_\mathbf{k}$ (of all joints in $K_1, ..., K_n$) and torques τ_p (of all joints in P) into

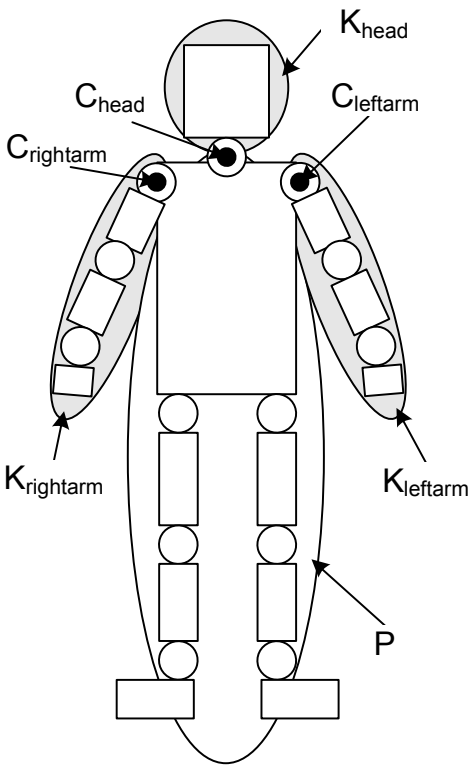


Figure 3.2: A body divided into kinematic parts that steer the arms and head and a physical part that steers the lower body and trunk.

account simultaneously. The equation of motion then has the form

$$\begin{bmatrix} \tau_k \\ \tau_p \end{bmatrix} = \mathbf{H}(\mathbf{q}) \begin{bmatrix} \ddot{\mathbf{q}}_k \\ \ddot{\mathbf{q}}_p \end{bmatrix} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{f}^x), \tag{3.1}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are vectors of generalized joint position, velocity and acceleration, $\mathbf{f}^x$ is a vector of external forces (including gravity), $\mathbf{H}$ is the joint-space inertia matrix and $\mathbf{C}$ is the joint-space bias force. $\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{f}^x$, $\mathbf{H}$ and $\mathbf{C}$ are considered inputs for the algorithm. Intuitively, $\ddot{\mathbf{q}}_\mathbf{p}$ and τ_k can be calculated if $\ddot{\mathbf{q}}_\mathbf{k}$ and τ_p are known.

Currently there is no real-time physics engine that solves the equation of motion for such a hybrid specification of joint torque and acceleration. Furthermore, solving the equation of motion given both forces and accelerations cannot be done using efficient iterative approaches such as the recursive Newton Euler approach [213]. The recursive Newton Euler approach has a complexity of $O(n)$, where n denotes the number of joints. Typically, in a hybrid system, the equations of motion are solved using a Lagrangian approach, which has a complexity of $O(n^3)$.

Because of this, I opted to sacrifice a slight amount of accuracy to gain calculation efficiency and allow my hybrid method to be used with current real-time physics engines. Rather than calculating the acceleration $\mathbf{a}_{Cj}$ of C_j, at the *current* frame, I use $\mathbf{a}_{C_j}'$, the acceleration of C_j at the *previous* frame to calculate the forces that each K_j exerts on P. Using this simplification, the movement of the K_j's can be modeled as movement of isolated systems, connected to a moving base C_j, that moves with

Table 3.1: Terms used in the Featherstone recursive Newton Euler approach

$\mu(i)$	set of children of body i
$\lambda(i)$	parent of body i
N_B	number of rigid bodies
$\mathbf{a}_g$	spatial gravitational acceleration
$\mathbf{v}_{C_j}$	spatial velocity of connector j
$\mathbf{a}'_{C_j}$	spatial acceleration of connector j at the previous frame
$\mathbf{v}_i$	spatial velocity of body i
$\mathbf{a}_i$	spatial acceleration of body i
$\mathbf{q}_i$	generalized DoF value vector of joint i
$\mathbf{S}_i(\mathbf{q}_i)$	matrix that maps generalized joint velocities on the DoF to spatial joint velocity
$\mathbf{I}_i$	spatial inertia tensor of body i
$\mathbf{f}_i^B$	spatial net force on body i
$\mathbf{f}_i^x$	spatial external force on body i
$\mathbf{f}_i$	spatial force transmitted across joint i
τ_i	torque exerted on joint i

acceleration $\mathbf{a}'_{C_j}$. The torque of each joint in each K_j can then be efficiently calculated using the recursive Newton Euler approach. The reactive torque of the parent joint in K_j is then applied to the rigid body in P that is connected to this parent joint. I make use of Featherstone's formulation of recursive Newton Euler approach, using 'spatial' 6-vectors [75]. The transformation from such spatial vectors to the traditional 3-vectors is shown in Appendix B. Table 3.1 summarizes the terms used in Featherstone's formulation of the recursive Newton Euler approach.

For the sake of clarity I model each K_j as a chain of joints. This is not a limitation of the system, as the recursive Newton Euler approach can easily deal with a branching tree of joints. K_j contains a chain of N_B rigid bodies, connected by $N_B - 1$ joints. An additional joint (joint 1) connects the chain to P at its connector location. The bodies are sequentially numbered $1..N_B$, starting with body 1, which is connected at the connector location C_j by means of joint 1. The remaining joints connect the rigid bodies in the chain: joint $i \in 2..N_B$ connects body $i-1$ with body i. Figure 3.3 illustrates the numbering convention used.

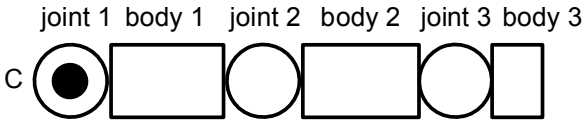


Figure 3.3: Numbering convention for joints and rigid bodies.

The spatial velocity of body i can be calculated as the sum of the spatial velocity of its parent and the spatial velocity across the joint connecting it to its parent:

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{S}_i \dot{\mathbf{q}}_i \qquad (\mathbf{v}_0 = \mathbf{v}_{C_j}), \tag{3.2}$$

where $\lambda(i)$ denotes the number of the parent of joint i. $\dot{\mathbf{q}}_i$ is the n-dimensional vector of generalized joint velocity, in which n is the number of degrees of freedom of the joint. $\mathbf{S}_i$ is a $6\times$n matrix that maps $\dot{\mathbf{q}}_i$ to spatial joint velocity. The spatial acceleration of body i can be calculated by differentiating equation 3.2:

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \mathbf{S}_i\ddot{\mathbf{q}}_i + \dot{\mathbf{S}}_i\dot{\mathbf{q}}_i \qquad (\mathbf{a}_0 = \mathbf{a}'_{C_j} + \mathbf{a}_g) \tag{3.3}$$

The net force acting on body i is given by the equation of motion

$$\mathbf{f}_i^B = \mathbf{I}_i\mathbf{a}_i + \mathbf{v}_i \times^* \mathbf{I}_i\mathbf{v}_i \tag{3.4}$$

in which $\mathbf{I}_i$ is the 6×6 spatial inertia tensor. $\times^*$ is the spatial cross product of force and velocity operator (see equation B.5 in Appendix B). Successive iteration of equations 3.2, 3.3 and 3.4 with i ranging from 1 to N_B provides the net forces acting on all bodies in the chain.

The spatial force transmitted from body $\lambda(i)$ to body i, across joint i is given by:

$$\mathbf{f}_i = \mathbf{f}_i^B - \mathbf{f}_i^x + \sum_{k\in\mu(i)} \mathbf{f}_k, \tag{3.5}$$

in which $\mu(i)$ is the set of children of a body. For a chain of bodies

$$\mu(i) = \begin{cases} \emptyset & \text{if } i = N_B \\ \{i+1\} & \text{if } i < N_B \end{cases} \tag{3.6}$$

$\mathbf{f}_i^x$ is the net external spatial force acting on body i. The values of such external forces are assumed to be known. For instance, gravity can be modeled as an external spatial force.[2] Figure 3.4 illustrates equation 3.5 for a chain of rigid bodies.

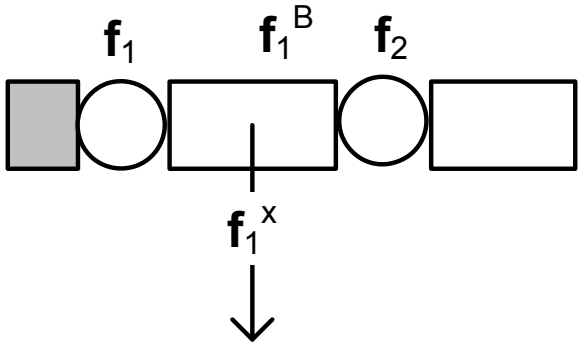


Figure 3.4: Spatial forces acting on rigid body 1. $\mathbf{f}_1^B = \mathbf{f}_1 + \mathbf{f}_1^x - \mathbf{f}_2$, in which $-\mathbf{f}_2$ is the reactive force of joint 2 on body 1.

Successive iterations of equation 3.5 with i ranging from N_B down to 1 will calculate the spatial forces acting on all joints in the chain.

[2]However, it is more efficient to model a uniform gravitational field as a fictitious spatial acceleration of C_j, as I did using the gravitational acceleration vector $\mathbf{a}_g$ in equation 3.3.

Finally, the torque at joint i is given by:

$$\tau_i = \mathbf{S}_i^T \mathbf{f}_i, \tag{3.7}$$

The reactive torque $-\tau_1$ is the torque exerted by K_j on a physical body consisting of P and K_j. I assume that the inertia of K_j is small compared to the inertia of P and apply a reactive torque $-k\tau_1$ directly to the rigid body from P that is connected to K_j. Alternatively, one can augment the inertia of P with the current combined inertia of the rigid bodies in K_j, by modifying the inertia tensor of rigid body from P that is connected to K_j on each simulation frame if the physical simulator allows one to do this. If this is not the case, one could use the articulated body method ([75], chapter 7) to calculate the spatial acceleration of a physical body P augmented with K_j resulting from applying $-\tau_1$. The torque to be applied on a physical body consisting solely of P (as used in the simulator) can then be calculated to achieve this desired spatial acceleration. In practice the assumption holds for kinematic gesture motion on the arms and neck combined with physical motion on the lower body and physically convincing motion is generated without requiring such computations.

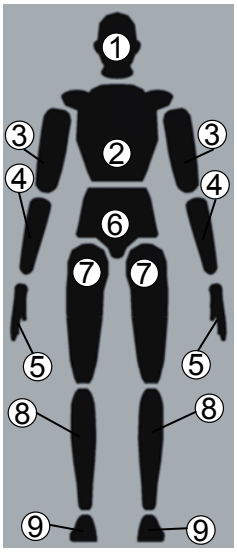
For a value of $k = 1$, the exact torque generated by the kinematic chain is applied to the rigid body in P. Values of k in the range $0 < k < 1$ can be used to increase the stability of the physical simulation. This can be seen as a crude way to model an increase in muscle tension to dampen the effect of large movements. Values of $k > 1$ can be used to exaggerated the effect of the joint torques of the kinematic motion.

3.2 Mixed Dynamics In Practice

I illustrate the use of my mixed kinematic/physical simulation system by combining a physical balancing model for the lower body with kinematic motion: a procedural arm swing, conducting arm gesture, a speech-accompanying gesture or a motion capture recording. Videos of these animations are available at `http://thesis.herwinvanwelbergen.nl/`.

3.2.1 Constructing a Physical Model of the Virtual Human

The physical model of the virtual human used in the examples in this chapter consists of 15 rigid bodies, connected by 14 joints (see Figure 3.5). Meshes of these rigid bodies were constructed by segmenting the mesh of the original virtual human, adapting it to be skintight, and closing the gaps in the resulting segments. I assume that the rigid bodies have a uniform density ρ. This density can be measured directly, from cadavers for instance, or using scanning systems that produce the cross-sectional image at many intervals across the segments [307]. I use the density table from [307], which provides densities for all segments but the sacroiliac, where I use the density given in [67]. Given the uniform density of each body and its closed polyhedral shape, its mass, its center of mass and its inertia tensor can be determined using [195]. The results are shown in Figure 3.5.



	body part	ρ kg/l	mass kg	center of mass (x,y,z) in m		
1	head&neck	1.11	4.80	0.00	1.63	0.01
2	upper-body	1.03	25.20	0.00	1.28	0.02
3	upper arm	1.07	2.95	±0.31	1.44	-0.01
4	lower arm	1.13	1.59	±0.32	1.17	-0.03
5	hand	1.16	0.70	±0.27	0.88	-0.01
6	sacroiliac	1.03	10.56	0.00	1.00	0.01
7	upper leg	1.05	8.79	±0.09	0.77	0.01
8	lower leg	1.09	3.73	±0.10	0.34	-0.03
9	foot	1.10	1.41	±0.10	0.04	0.02
	Total		94.21			

Figure 3.5: Segmentation of the virtual human into rigid bodies and the inertial properties of the bodies

I base the joint rotation limits for the physically steered joints on data from male US air force personal [310](see table 3.2).

Precise collision shapes are typically not crucial in Elckerlyc's applications, and collision detection is fast when simple bounding shapes, such as boxes, capsules and spheres are used. In the examples in this chapter, I set the collision shape of the rigid bodies to the bounding box of their mesh. If more precise collision detection is needed, the actual mesh of the rigid body can be used as a collision shape, or the collision shape can be approximated with a combination of simple bounding shapes.

3.2.2 Obtaining Joint Velocity and Acceleration

The joint velocities and accelerations for each joint are calculated from their rotation data at time t, $t - h$ and $t + h$. I define $\mathbf{p}(t)$ as the rotation of a joint at time t. If the simulation rate is set to step size h, it is possible to reuse the $\mathbf{p}(t + h)$ and $\mathbf{p}(t)$ values from the previous simulation step, so that in each step only $\mathbf{p}(t + h)$ needs to be calculated. In the examples, h is 3 ms.

3.2.2.1 Reparameterization

The rotation of the joints is represented by quaternions. The quaternions $\mathbf{p}$ and $-\mathbf{p}$ represent the same rotation. For a sequence of quaternions, representing the rotation of a joint, switches between these alternate representations causes large differences between the quaternion components of quaternions that actually represent (nearly) the same rotation. This is undesired for signal analysis techniques that work on quaternion components, such as filtering and numerical differentiation. Therefore I reparameterize $\mathbf{p}(t)$ and $\mathbf{p}(t + h)$ so that the distance between the quaternion components of $\mathbf{p}(t - h)$ and $\mathbf{p}(t)$ and between $\mathbf{p}(t)$ and $\mathbf{p}(t + h)$ is

Table 3.2: Joint rotation limits, in degrees, in a right-handed coordinate system, with the y-axis pointing up and the virtual human facing the positive z direction. Rotation limits for the shoulder around the y-axis are not provided in [310].

joint	x_{min}	x_{max}	y_{min}	y_{max}	z_{min}	z_{max}
left wrist	-27	47	-	-	-90	81
right wrist	-27	47	-	-	-81	90
left forearm	-	-	-103	113	-	-
right forearm	-	-	-113	103	-	-
left elbow	-142	0	-	-	-	-
right elbow	-142	0	-	-	-	-
left shoulder	-188	61	?	?	-48	134
right shoulder	-188	61	?	?	-134	48
neck	-60	61	-79	79	-41	41
left ankle	-38	35	-	-	-24	23
right ankle	-38	35	-	-	-23	24
left lower leg	-	-	-43	35	-	-
right lower leg	-	-	-35	43	-	-
left knee	0	113	-	-	-	-
right knee	0	113	-	-	-	-
left hip	-113	0	-31	30	-31	53
right hip	-113	0	30	31	-53	31

minimized:

$$\tilde{\mathbf{p}}(t) = \begin{cases} -\mathbf{p}(t) & \text{if} \quad \mathbf{p}(t-h) \cdot \mathbf{p}(t) < 0 \\ \mathbf{p}(t) & \text{otherwise} \end{cases} \tag{3.8}$$

$$\tilde{\mathbf{p}}(t+h) = \begin{cases} -\mathbf{p}(t+h) & \text{if} \quad \tilde{\mathbf{p}}(t) \cdot \mathbf{p}(t+h) < 0 \\ \mathbf{p}(t+h) & \text{otherwise} \end{cases} \tag{3.9}$$

where $\tilde{\mathbf{p}}(\mathbf{t})$ is a reparameterized quaternion rotation at time t and $\mathbf{p}(t)$ is the original rotation at time t.

3.2.2.2 Filtering

Motion capture data contains high frequency noise. This noise gets amplified with time differentiation [307]. Noise will dominate the signal after double differentiation. To prevent this, I make use of the 2-pass Butterworth low pass filter proposed in [307] to cut off high frequency noise before differentiating. The filter is described by:

$$\tilde{x}_i = \begin{cases} x_i & \text{if} \quad i < 2 \\ a_0 x_i + a_1 x_{i-1} + a_2 x_{i-2} + b_1 \tilde{x}_{i-1} + b_2 \tilde{x}_{i-2} & \text{otherwise} \end{cases} \tag{3.10}$$

where $\tilde{x}_i$ is the filtered data at frame i and x_i is the raw data at frame i. In a

Butterworth filter, the filter coefficients a_0, a_1, a_2, b_1 and b_2 are calculated as follows:

$$\begin{aligned}
\omega_c &= \frac{\tan(\pi f_c / f_s)}{C} \\
a_0 &= \frac{K_2}{1 + K_1 + K_2}, \quad \text{with} \quad K_1 = \sqrt{2\omega_c}, \quad K_2 = \omega_c^2 \\
a_1 &= 2a_0 \\
a_2 &= a_0 \\
b_1 &= -2a_0 + K_3, \quad \text{with} \quad K_3 = \frac{2a_0}{K_2} \\
b_2 &= 1 - 2a_0 - K_3
\end{aligned} \tag{3.11}$$

where f_c is the desired cutoff frequency, f_s is the sample frequency. The digital filter introduces a phase shift in the output signal relative to the input signal. To cancel out this phase shift, the once-filtered data is filtered again in the reverse direction of time [307]. C is a correction factor for each additional pass of the Butterworth filter:

$$C = (2^{\frac{1}{n}} - 1)^{0.25} \tag{3.12}$$

qn is the number of filter passes. In my case, $C = 0.802$. The exact value of f_c is not very critical, values of around 15-25 Hz work well in practice. I filter the s, x, y and z components of the quaternions in the keyframe data separately and re-normalize the quaternions after filtering. If the quaternions are reparameterized according to equation 3.8, the renormalization only slightly adjusts the filtered quaternions in my mocap recordings. Procedural motion is typically already smooth by design and does not need filtering.

3.2.2.3 Calculating Angular Velocity and Angular Acceleration

For ease of calculation, I model all joints driven by kinematic motion as ball joints, that is: with three rotational degrees of freedom. If I choose a joint's angular velocity vector ω in the joint's own coordinate system as its velocity variable $\mathbf{q}_i$, $\mathbf{S}$ reduces to

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.13}$$

[75]. ω and $\dot{\omega}$ can be determined from the quaternion rotation $\mathbf{p}(t)$ and its derivatives:

$$\begin{bmatrix} 0 \\ \omega \end{bmatrix} = 2\dot{\mathbf{p}}(t)\mathbf{p}(t)^{-1} \tag{3.14}$$

$$\begin{bmatrix} s \\ \dot{\omega} \end{bmatrix} = 2\ddot{\mathbf{p}}(t)\mathbf{p}(t)^{-1} \tag{3.15}$$

$\dot{\mathbf{p}}(t)$ and $\ddot{\mathbf{p}}(t)$ are determined using numerical differentiation of the reparameterized and optionally filtered joint rotations $\mathbf{p}(t-h)$, $\mathbf{p}(t)$ and $\mathbf{p}(t+h)$:

$$\dot{\mathbf{p}}(t) = \frac{\mathbf{p}(t+h) - \mathbf{p}(t-h)}{2h} \tag{3.16}$$

$$\ddot{\mathbf{p}}(t) = \frac{\mathbf{p}(t+h) - 2\mathbf{p}(t) + \mathbf{p}(t-h)}{h^2} \tag{3.17}$$

3.2.3 Simulation Details

The Open Dynamics Engine (ODE) [273] is used to generate the motion of the lower body, based on the joint torques provided by the balance controller and the torques calculated by inverse dynamics. It also handles the collision detection and contact of the physical model of the lower body with the floor. Friction of the feet with the floor is handled using ODE's simplification of Coulomb friction. In very long simulations (longer than 1 hour), small foot-lifts and accumulated simulation errors can slightly move the feet over time. If extra stability or calculation speed is needed, friction handling can be omitted by setting foot constraints that effectively 'glue' the feet to the floor, preventing them from moving completely. To take advantage of multi-processor systems, the physical simulation runs in a separate thread.

3.2.4 Balancing Controller

I use the balancing controller described in [312]. This controller dampens the velocity of the center of mass and steers it toward its desired position, specified by a predefined hip height and a horizontal balance location which lies in between the feet. The output of the controller are the torques, to be applied to hips, knees and ankles. To adapt to a body with different inertial properties, a single stiffness multiplier is used on all spring gains in the PD-controllers used in the balance controller. An estimation of the value of this stiffness multiplier can be calculated (see [114]), but in practice it's easier to tweak it manually. A video of the balance controller, using virtual humans with different physical properties (fat vs thin and tall) is shown at `http://thesis.herwinvanwelbergen.nl/`.

3.2.5 Results

Figure 3.6 shows a series of captured frames of animation generated with my system, using a combination of my physical balance model with a procedurally generated large arm swing. The motion enhancement created by my system is subtle for smaller kinematic motions and therefore hard to capture on a series of images. I refer the interested viewer to the videos at `http://thesis.herwinvanwelbergen.nl/` to see the system in action with more subtle kinematic motions, including several procedural conducting and other gestures and motion captured arm and head movements. I also reproduce one of the motions described in [123]: a physical swing is put into motion with a kinematically moving body.

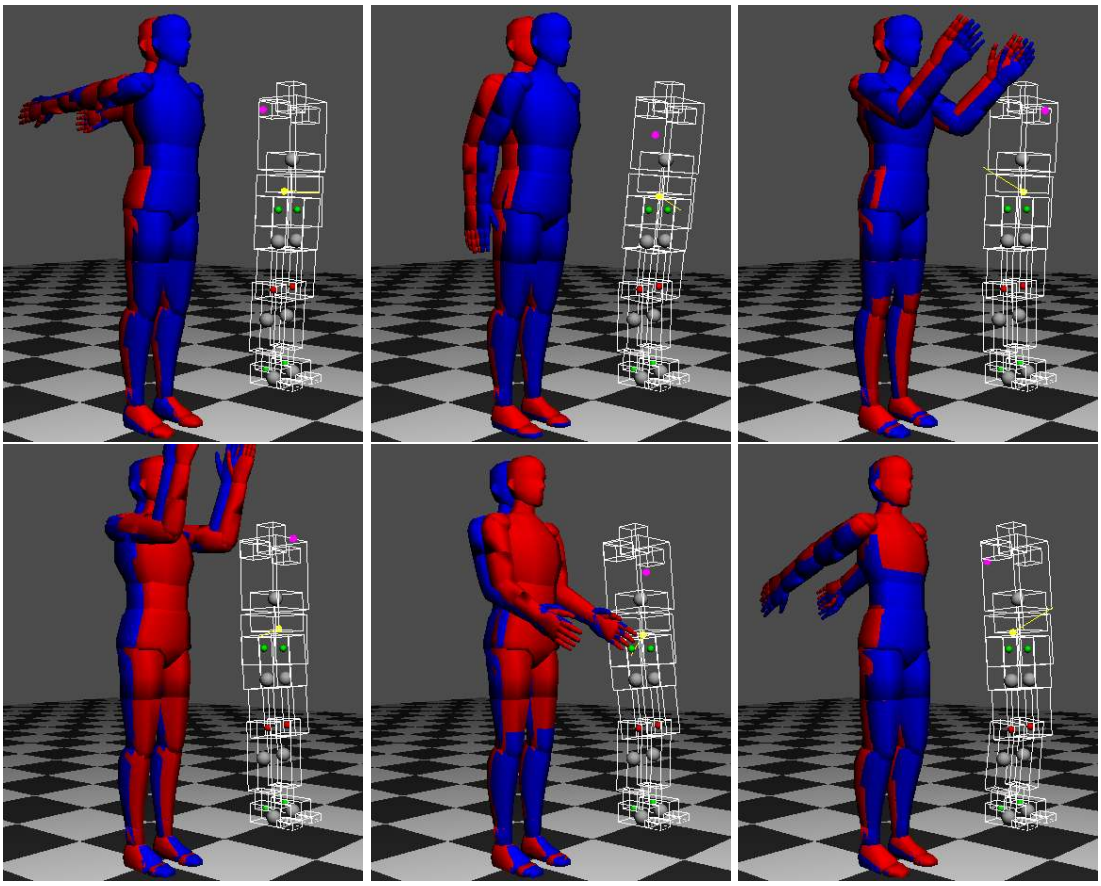


Figure 3.6: Mixing a kinematic arm swing with physical balancing. The blue virtual human is animated with physical simulation and kinematic arm motion, the red virtual human is animated solely with kinematic motion. The wireframe on the right side of each picture shows the visualization of the physical model of the lower body of the virtual human.

3.2.6 Performance

In a performance test, my system animated up to 30 conducting virtual humans in real time on a desktop computer (2.83 GHz, Quad core, 4Gb ram, Nvidia GeForce 8800 GTS video card). Each conductor is animated by its own procedural animation model and physical balance model. The physical simulation frame rate is set to 200 fps and the visual frame rate is around 50 fps. Roughly half of the simulation time is spent on the procedural animation, the other half on physical simulation. The video of this performance test is shown at `http://thesis.herwinvanwelbergen.nl/`.

Like my method, *motion tracking* can be used with any existing physics engine. Motion tracking (see Chapter 2.4.1.3 for an overview of techniques) uses physical simulation on the whole body. A tracking controller is used to compute the torque on each joint. The desired state of this controller is the desired rotation of the joint, as specified in motion capture or other kinematic data. Motion capture noise, tracking errors and environmental changes can easily disturb the balance of a character whose body is animated using a tracking controller. Therefore an explicit physical balance controller is still needed. Because tracking makes use of physical controllers, the motion generated by tracking has a time-lag relative to the kinematically specified motion and it is not guaranteed that the joint rotations specified in the mocap data are actually achieved. This makes tracking unsuitable for applications where precise timing and limb placement is needed.

My method potentially preserves the characteristics of the kinematic motion better than tracking methods. Furthermore, my method is far more efficient than tracking methods, not only because solving the equations of motion in my hybrid system is more efficient ($O((n-k)^3+k)$ vs $O(n^3)$ using ODE), but also because it avoids the expensive double integration of acceleration for the kinematically steered joints and does not need to do collision detection on those joints. A tracking method would be preferred over my method if realistic collision detection and response on kinematically steered joints is needed and precise timing and limb placement is less crucial.

Unlike methods that model the physical balancing solely through the displacement and velocity of the center of mass [201, 211], my method also models the force transference from the arms to the trunk. This results in a more natural 'sharper', less smooth movement of the lower body when large accelerations occur in arm and head movement. The videos at `http://thesis.herwinvanwelbergen.nl/` illustrate this with a clapping motion and several conducting motions.

3.3 Discussion

I have developed a system that can combine kinematic motion with physical simulation in a physically coherent manner. The balance controller used in my system is relatively simple. What I did not model yet is the fact that human balancing is not a purely *reactive* process. The balance controller therefore lacks the notion of anticipation. For example, it does not move backward *in advance* to anticipate a

large arm swing forward, as a real human might do. This is not a limitation of the mixed dynamics system itself, another more elaborate balance controller could be designed that takes this into account.

The inverse dynamics analysis of kinematic movement does not only yield the reactive torque, but also the torque on all other kinematically steered joints. This type of information can potentially be used in the motion planning stage, for example to drop a load if it is too heavy, or to show an angry facial expression when some motion costs more effort than anticipated.

Other hybrid physical simulation/kinematic systems [262, 325, 326] have been designed to switch between full-body kinematic motion and full-body physical simulation, depending on the current situation's needs. Such systems can show realistic interaction with the environment (e.g. falling) when needed. Rather than doing full-body switches, I allow switching to a different mix of physically and kinematically steered joints in real time. One of the usage scenarios for this is the virtual conductor (see also Chapter 8). A conductor typically conducts with his right hand and uses the left hand only for expressive cues. If the left hand is not needed, it should hang down loosely. I modeled this loose movement using a simple PD pose controller (see the movie on the web page of this thesis). The desired state for the controller is the desired rotation of the shoulder and elbow joints. The animation needed to create the expressive left hand cues require tight synchronization to the music and is therefore generated by procedural motion. Switching from loosely hanging arm movement to expressive left hand conducting gesture and back occurs in real time and requires switching between different mixes of physically and kinematically steered joints. A switch from kinematical to physical control on K_j is implemented by augmenting P with the rigid body representation of K_j and applying the current joint velocity and rotation to the matching joints in the new physical representation. This will obviously result a similar torque being executed on P. Therefore such a switch results in smooth movement. A switch from the physical to kinematic control removes the physical representation of a body part from P and inserts a new kinematic chain K_j. To ensure that no sudden torques occur on the new physical body, the movement on K_j directly after the switch must be similar to the movement in its former physical representation. Chapter 8.7 discusses the animation plan requirements for such switching and its setup in Elckerlyc in more detail.

Chapter 4

The Motor Plan

The motor plan of a virtual human describes how it achieves some intentional goal (e.g. walk to a door and open it, inform a student that he needs to work harder) using a set of coordinated PlanUnits. In addition to that, it may contain PlanUnits that reflect the virtual human's unconscious behavior (blinking, breathing, etc.).

MotionUnits are a specific category of such PlanUnits that execute computer animation. In the previous chapters, I have discussed how several animation techniques can be employed to generate the motion for virtual humans. Here I show how several state-of-the-art animation techniques are implemented in the MotionUnits of my virtual human platform Elckerlyc.

The PlanUnits in a motor plan are tightly coupled to each other in both timing and shape. In the previous chapter I have described the mechanical coupling between PlanUnits; this chapter shows that, in addition to this mechanical coupling, PlanUnits are also tightly coordinated through neurological coupling processes.

The coordination of the PlanUnits in the motor plan thus reflects the ordering of PlanUnits needed to achieve some intentional goal or execute some reactive behavior and the constraints that satisfy the neurological coupling between the PlanUnits. This chapter illustrates several such neurological couplings. In Chapter 5, I show that the constraints that describe the coordination between PlanUnits within one's own body are very similar to the constraints that describe our coordination with others. In virtual human applications, the motor plan is typically described by a multimodal specification language. Such languages (see Chapter 6.1 for a historical overview) describe the coordination between PlanUnits as constraints between their key time moments (keys).

4.1 PlanUnits: Elements of Motor Movement

I model the execution of motor movement (including speech) as the coordinated execution of PlanUnits that form a multimodal motor plan. A similar modular organization of motion plans is found in neuroscience [31, 276] and most computer animation approaches (among many others: [96, 104, 152, 203, 218, 312]).

Each PlanUnit has a predefined semantic function (for instance: a three-beat

conducting gesture, a speech clause, an eyeblink). It has at least a certain start time and may contain several other key time moments (further called *keys*) that are relevant for its coordination with other PlanUnits or events in the (virtual) world. Some PlanUnits represent discrete actions that have a clear end (for example: a speech clause, a gesture). Others represent ongoing behaviors (for example: standing in a balanced pose, letting an arm hang down loosely). PlanUnits can represent behavior that is executed ballistically, or behavior that is continuously adapted on the basis of perceptual or other feedback. To allow the latter, the timing and parameterization of ongoing PlanUnits can be updated continuously. Thus, PlanUnits can be seen as a mapping f from current time t and a set of parameter values $\mathbf{a}$ to a set of control primitives (for example: a pose in animation) $\mathbf{c}$.

$$f(t, \mathbf{a}) = \mathbf{c} \tag{4.1}$$

To allow parameters to be used over different embodiments of a virtual human, they can be defined in units relative to the embodiment. For example, a pelvis height parameter in a balance controller is better defined as a percentage of leg length than in an absolute value in meters. The MPEG-4 Facial Animation standard [215] achieves face independent parameterization of animation in this manner: its animation is specified in specific measurement units, called Facial Animation Parameter Units, which represent fractions of key facial distances (e.g. the distance between the eyes, the mouth width, etc.).

Furthermore, parameters should be independent of execution channel. For example, speech volume can be defined in percentage rather than decibel so that it can be used in both text synthesis (mapping to font size) and speech synthesis (mapping to audio volume).

4.2 MotionUnits: the PlanUnits of Animation

MotionUnits[1] form the specific category of PlanUnits that steer the motion of a virtual human. Elckerlyc uses two types of MotionUnits: kinematical MotionUnits steer the virtual human through rotations and translation of joints in its skeleton, and physical MotionUnits that use torques and forces to steer the physical representation of the virtual human. Each MotionUnit acts on a selected set of joints.

In Chapter 2.2 I defined a motion primitive as the mapping of time to the DoF values of a skeleton. A motion space was defined as the collection of motion primitives with the same semantic function. A MotionUnit is a continuous mapping from both time *and* parameter values to the DoF of a skeleton and has a specific semantic function. Unlike a motion primitive which has a fixed path of DoF values it follows while being executed, a MotionUnit has the inherent ability to change its motion on the basis of parameter value changes or timing changes while it is being executed.

[1] Motion structures that have a function and granularity that is similar to Elckerlyc's MotionUnit have been called gestures [104, 209], controllers [280, 312], verbs [240], motion units [241], local motor programs [155, 319], atomic animated actions [218], clips [96], action units [267], or actions [107, 203, 217] in some other literature on virtual humans or computer animation.

Once the MotionUnit is fully executed, the path of DoF values it created can be seen as a motion primitive. The different motion primitives that can be created by a MotionUnit thus form a motion space whose size is dependent on the number of parameters the MotionUnit supports and their range.

Like the clips defined by Grassia [96], MotionUnits are executed on the basis of a canonical time value α, rather than directly on absolute time. Grassia measures canonical time in integer key frame numbers. Because most of Elckerlyc's MotionUnits are constructed using procedural motion models rather than keyframe animation, canonical time α is represented using a value between 0 and 1 instead ($\alpha = 0$ refers to the start of the motion, $\alpha = 1$ to its end).

MotionUnits contain one or more *motion phases*, separated by *keys*. Each key is assigned a predefined canonical time value $0 \leq \alpha_i \leq 1$ that indicates where it is located within the MotionUnit (See Figure 4.1 for some typical phases and keys for a gesture MotionUnit).

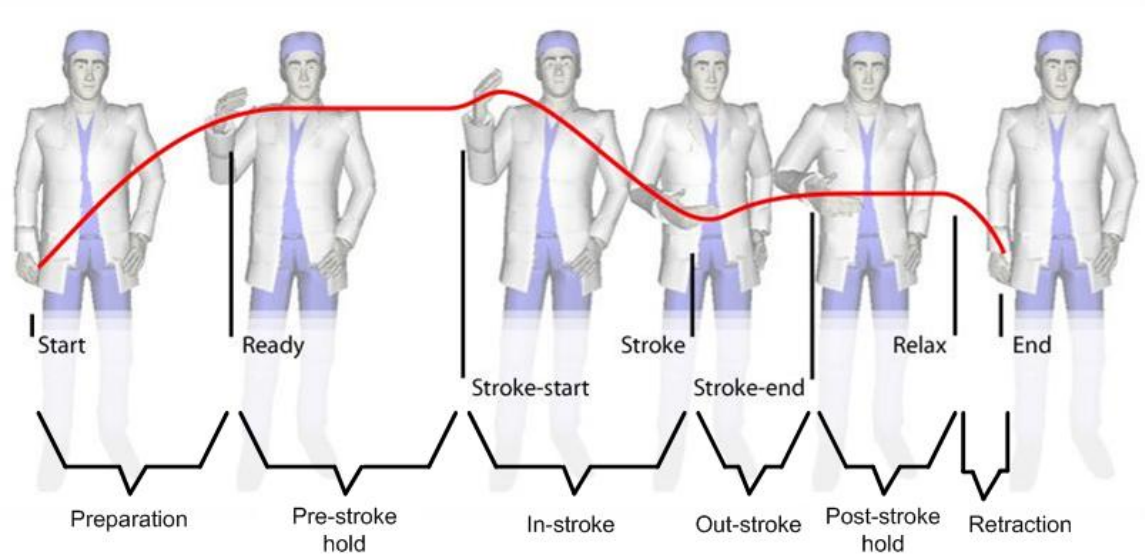


Figure 4.1: Typical typical phases and keys for a gesture MotionUnit (picture from `http://wiki.mindmakers.org/projects:bml:main`).

Given the current set of parameter values $\mathbf{a}$ and a canonical time $0 \leq \alpha \leq 1$, a MotionUnit can be executed, typically by rotating some joints of the virtual human. I employ a time warping technique to set up the mapping from absolute time to α (see also Chapter 8.7.5).

4.2.1 Procedural MotionUnits

Procedural MotionUnits rotate joints over time as specified by mathematical expressions that take α as well as a vector $\mathbf{a} \in \Re^n$ as parameters. These expressions can

be used to steer joint rotation directly, to position the skeletal root or to position the wrists and ankles using analytical inverse kinematics [287]. The procedural MotionUnits support the specification of wrist/ankle positions as continuous mathematical formulas in both global (world) and local (shoulder) coordinate systems. For example, the expression

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

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

Joint rotations can be specified as continuous mathematical functions of α and $\mathbf{a}$, or as global or local rotation values defined procedurally (as a function of $\mathbf{a}$) at key times (in a similar manner as in [104, 111]).

The parameter values $\mathbf{a}$ can be changed in real time, changing the motion shape or timing. All mathematical expressions are evaluated using the Java Math Expression Parser.[2] Custom function macros can be designed. I have defined such macros for Hermite splines, TCB splines [146] and Perlin noise. Additional examples of XML-specifications of procedural MotionUnits can be found on the web page of this thesis.

This design — allowing arbitrary mathematical formulas and parameter sets to be used for motion specification — is more flexible than traditional procedural animation models that define motion in terms of splines or other *predefined* motion formulas and use *fixed* parameter sets [58, 104, 111]. Since it is compatible with these traditional methods, Elckerlyc can make use of such existing procedural animations. I have semi-automatically converted several MotionUnits from Greta [104] into the XML description for procedural animation. Motion capture animation is also incorporated as a procedural MotionUnit.

4.2.2 Custom Programmed Procedural MotionUnits.

While this generic procedural motion definition in XML is very flexible, it is sometimes more convenient to author procedural MotionUnits by programming them directly. By doing this, the motion author gains direct access to functionality within Elckerlyc's AnimationPlayer. This functionality includes the prediction of a pose at a given time (see Section 4.2.4) and provision of the current joint pose of the virtual human, which makes it very easy to author a MotionUnit with a flexible start and/or end pose. Several such Custom MotionUnits have been implemented in Elckerlyc.

The Gaze MotionUnit The Gaze MotionUnit that steers the head and eyes is implemented on the basis of Tweed's biological model of gaze [293]. This model provides a comfortable rotation of the head given a certain gaze direction. Because the head usually moves more horizontally than vertically, the model scales horizontal and vertical components of the desired head rotation differently. The torsional component of the head rotation is scaled to fit Donders' law of the head [225]. The

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

desired eye rotation is recalculated for each frame, given the current head rotation. This models the eye overshoot that is observed in gaze. That is, the eye can lock onto the gaze target before the head does and then glide 'back' to achieve its final position as the head catches up.

Tweed's model also provides head and eye velocity profiles, which are currently not implemented in the Gaze MotionUnit. Instead, the Gaze MotionUnit uses a simple bell shaped velocity profile.

The rotation axis of the head during gaze movements is approximately constant throughout the movement [294]. The gaze MotionUnit limits the eye gaze rotation to be within the biologically motivated rotation limits (obtained from [293]).

Saccades Saccades are quick, simultaneous movements of both eyes in the same direction, used (among other things) to shift gaze. The duration of a saccade is linearly dependent on its amplitude (in radian) [45]. Saccades have a symmetric velocity profile. The peak velocity of the saccade is linearly dependent on its amplitude (with a plateau velocity of around 8.7 radian / s) [45].

Elckerlyc's eye-only gaze MotionUnit adheres to the duration rule specified in [45]. Its velocity is currently set as a constant rather than the more biomechanically correct symmetric peak described above. Again, the eye rotations are limited to be within biologically motivated rotation limits.

The Pointing MotionUnit Pointing gestures are implemented using a custom MotionUnit that moves the pointing arm from its start pose to a pointing target, keeps it there during the stroke phase, and then moves it back to the starting pose. The pointing gesture has a symmetric retraction and preparation movement, motivated by similar symmetry in pointing observed in humans [140, 303].

Many studies have shown that the hand trajectory for reaching and pointing movements has a bell-shaped velocity profile [320]. A clear acceleratory and deceleratory phase can be recognized. This bell is usually asymmetric, that is, the length of the acceleratory phase can be different from that of the deceleratory phase. The Pointing MotionUnit provides a custom, configurable sigmoid function that describes the relative position-time diagram of the arm position. This sigmoid allows one to adjust its steepness and the length of the acceleratory and deceleratory phases. It thus achieves an adjustable bell shaped velocity profile. The exact implementation is detailed in [303].

4.2.3 Physical MotionUnits

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

such controllers can cope with, and recover from, external perturbations. I have implemented several physical controllers.

Pose Controllers Simple proportional derivative controllers are used as pose controllers that loosely keep a body part in its desired position, while still being affected by forces acting on the body. The pose controller acts upon a selected hinge or ball joint.

Balance Controller I have implemented the balance controller described in [312]. This controller dampens the velocity of the center of mass and steers it toward its desired position, specified by a predefined hip height and a horizontal balance location which lies in between the feet. The output of the controller are the torques, to be applied to hips, knees and ankles.

Rag doll Controller The rag doll controller is a controller that acts upon all joints of the physical body of the virtual human it steers. It applies no torques to any of these joints. This controller makes the virtual human collapse like a rag doll, an effect that can be used to simulate movement during heavy collisions (e.g. being hit by a car) or to simulate death animations.

Compound Controllers A compound controller combines several controllers into a single controller. This combination is described in an XML specification file. Figure 4.2 shows an example of such a compound controller: a controller to let the left arm hang down loosely is composed of three PD controllers controlling the left shoulder, left elbow and left wrist joint respectively. Compound controllers are composed of a set of *required* controllers that are essential for their functioning and a set of *desired* controllers that should be enabled if the physical representation of the virtual human allows it (e.g. it contains the joints steered by the desired controller). For example, the loosely hanging left arm controller of Figure 4.2 only dampens the movement of the left wrist if it is available in the physical body.

4.2.4 Transition MotionUnits

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

```
<CompoundController xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
  <required>
    <Controller class="hmi.physics.controller.BallJointController"
     id="shoulder">
      <bmlt:parameter name="joint" value="l_shoulder"/>
    </Controller>
    <Controller class="hmi.physics.controller.HingeJointController"
      id="elbow">
      <bmlt:parameter name="joint" value="l_elbow"/>
    </Controller>
  </required>
  <desired>
    <Controller class="hmi.physics.controller.HingeJointController"
     id="wristx">
      <bmlt:parameter name="joint" value="l_wrist"/>
      <bmlt:parameter name="axis" value="0"/>
    </Controller>
    <Controller class="hmi.physics.controller.HingeJointController"
     id="wristz">
      <bmlt:parameter name="joint"  value="l_wrist"/>
      <bmlt:parameter name="axis" value="2"/>
    </Controller>
  </desired>
</CompoundController>
```

Figure 4.2: A compound controller specification for a loosely hanging left arm.

I have designed a transition MotionUnit based upon a slerp transition on each joint and one that creates a C^2 continuous rotation curve between joint rotations [143].

4.3 Intrapersonal Multimodal Synchrony

The movement of the body should not be seen as a process of executing a set of completely independent PlanUnits steering separate body parts. The PlanUnits are tightly coupled. In Chapters 2 and 3, I discuss models for coordination in the form of mechanical coupling between limbs (e.g. through force transference between body segments). This section discusses the coordination between PlanUnits (including the coordination of motion with speech) at a neurological level.

4.3.1 Inter and Intra-limb Synchronization

Periodic bimanual movements are often the focus of studies on basic organization of human actions [102, 133, 134, 191, 237]. A common finding in all these studies is that only two patterns of rhythmic bimanual coordination can be achieved without training: a stable 'in-phase' pattern and a less stable 'anti-phase' pattern. At higher movement frequencies, the stability of the anti-phase patterns decreases,

eventually resulting in a loss of stability, typically followed by an involuntary transition to the in-phase pattern [133]. When the frequency is then reduced again, the movement does not automatically return to an anti-phase pattern [102]. Similar coordination is observed in the organization of the movement of the elbow and wrist joints within a single limb [135] and between synchronized movement of the knee and elbow [136]. The spatial orientation of the body also determines which pattern is stable (and thus called in-phase). The synchronized flexing/extending of homologous muscles may be an in-phase pattern in one body pose and the anti-phase pattern in another [135, 136, 191]. When the two coupled components are not equivalent (e.g. in mass), absolute phase and frequency synchronization may no longer be observed; only tendencies for in-phase and anti-phase coordination are present, interspersed with desynchronization and phase wandering [136]. Typically, two stable states still exists that are close to (but not exactly at) either a 0° or 180° phase difference between the components. Again, the anti-phase pattern is less stable. At higher movement frequencies its stability decreases and involuntary transitions to the in-phase pattern occur. In such asymmetric systems, fixed point drifts (of the stable phase) are observed with frequency changes. Treffner and Turvey [291] show that, even when the coupled components are equivalent, slight differences in phase can occur between, for example, the left and the right hand. Right-handed individuals typically 'lead' the movement with their right hand and left-handers with their left. This phase difference between hands increases with movement frequency.

Haken, Kelso and Bunz [102] propose a model of two coupled oscillators that fit the observations of symmetric limb coordination (the HKB-model). This model was later extended to fit the synchronization observations of inequivalent limbs [136]. Treffner and Turvey [291] provide an asymmetric extension of the HKB model which provides a small anisotropic coupling element to fit the observations on handedness. The exact channel of the coupling between the oscillators is not given in these models. Ridderikhoff et al. [237] show that a combination of several interlimb interactions underlie the stability characteristics of rhythmic interlimb coordination: the integrated timing of feed-forward control signals, phase entrainment through contralateral afference (=reception of sensory signals) and timing corrections based on the perceived error of relative phase.

Tight synchronization also occurs between discrete (rather than rhythmic) actions: Kelso [137] shows that when subjects have to point at an easy and a hard target simultaneously with two hands, the movement of the hands is tightly coordinated. That is, the timing and the velocity profile of the hand movement of the easy task adjusts to that of the hard task.

Adamovich et al. [3] show some interaction effects between rhythmic and discrete arm movements of the same arm. Subjects perform a rhythmic elbow movement around a target and are instructed to move it to another target upon a trigger. The initiation of the discrete movement to the new target resets the phase of the rhythmic movement. The onset of the discrete movement was confined to a limited phase window in the rhythmic cycle. Sternad et al. [276] replicated these findings and show that the movement duration of the discrete movement was influenced by

the period of the oscillation.

4.3.2 Speech-Gesture Synchronization

Treffner and Peter [290] argue that since English speech seems rhythmical at the level of stressed syllables, it could potentially be coordinated with rhythmic movement. In an experiment they let subjects tap their index finger to the rhythm of a metronome, while saying /ba/. The subjects were asked to move either in-phase (synchronize jaw maximum down with finger maximum down) or in anti-phase (synchronize jaw closed with finger maximum down). The subjects were able to maintain both the in-phase and anti-phase coordination for all frequencies. The anti-phase coordination was less stable, and relations between frequency and phase were shown to match the asymmetric extension of the HKB model.

Others have looked at speech/gesture coordination in more natural settings. Condon and Ogston [60] observed —using micro-analysis of video images and speech— that the morpheme, syllable and word boundaries of speech are in alignment with the points in which the movement of limbs, head, eyes, eyebrows and mouth changes direction.

Later research provided further insights into the exact nature of this synchronization and revealed synchronization between speech and gesture at higher (e.g. locution, locution group and discourse) levels [139, 176, 189]. Gestures are hierarchically organized in a way similar to the organization of speech [139]. Kendon [139] observed that the elements of a similar 'level' within these two hierarchies are strongly synchronized (see Figure 4.3).

I provide a brief overview of some of these synchronizations here, the reader is referred to [176] for an extensive overview of both the synchronizations and synchronization mechanisms.

The left side of Figure 4.3 shows the organization of gesture used in this thesis. A *gesture unit* is defined as the period of time between successive rests of the limbs. A gesture unit begins when a limb starts to move, and ends when it has reached its resting position again. A gesture unit can contain several *gesture phrases*. The gesture phrase consists of one or more movement phases:

- preparation (optional), in which the limb moves away from the resting position to a position in gesture space where the stroke begins.
  - pre-stroke hold (optional) is the position and hand posture reached at the end of the preparation itself. This may be held until the stroke itself begins. Pre-stroke holds occur if for some reason the stroke onset is delayed [189].
- the stroke (obligatory) is the peak of effort in a gesture. In this phase, the meaning of the gesture is expressed.
  - post-stroke hold (optional) is the final position and posture the hand reaches after a stroke. This may be held until the retraction begins. Post-

> stroke holds occur if, for some reason, the co-expressive spoken utterance is delayed [189].

- retraction (optional) is the return of the hand to a rest position.

Gestural units can be grouped by a common feature, typically a consistent head movement pattern, that occurs in all of them. At the highest level, Kendon noted consistencies in arm use (left, right, both) and body posture.

A speech stream can be segmented into intonation tune units or *tone units*. A tone unit is a group of syllables over which there is a complete intonation tune (e.g. rise-fall, so roughly a clause or a sentence). The gestural stroke typically occurs at or slightly before the stressed syllable in such a tone unit ([82, 139, 189], but challenged by [243]). Loehr [176] provides a more detailed account of this synchrony. He observed that the *apex* of the stroke of a gesture tends to align with a pitch accent in speech.

Tone units contain *intermediate phrases*. An intermediate phrase is defined by an intonation contour with one or more pitch accents and a phrase accent, but no (final) boundary tone. Gesture phrases align with intermediate phrases [176]. The gesture phrase typically starts and ends slightly before the intermediate phrase (on average with 100ms). Typically there is a one to one alignment, but often multiple gesture phrases align with one intermediate phase. The reverse (the occurrence of multiple intermediate phrases within one gesture phase) occurs seldom.

Tone units combine into groups called *locutions*. Locutions are usually complete sentences. Kendon observed that all locutions have their own gesture unit. That is: the boundaries of the locution are associated with the gesticulatory limb either being in the rest position or returning to the rest position.

Locutions combine into *locution groups*; that is, locutions sharing a common intonational feature apart from other groups of locutions, for example they might all end with low-rise. Consistent head movement patterns are typically observed over all locutions within a locution group. Locution groups combine into *locution clusters*.

Locution clusters can be seen as discourse paragraphs. They are separated by a pause or a marked change in voice quality, loudness or pitch range. During a locution cluster, speakers often consistently gesture with the left, right or both hands [139, 190].

Locution clusters combine into a *discourse*, which is equivalent with a speaker's turn. Kendon observed that speakers sustain a certain body posture that contrasts with the posture before or after the discourse.

Cassell et al. [49] show that posture shifts occur frequently at both locution cluster (which they call discourse segments) boundaries and discourse boundaries.

Loehr [176] notes that eye blinking synchronizes with gesture and speech as well:

> I found that eye blinks typically happen on the rhythmic pulse. A casual viewing of my video data (or of anyone speaking) will confirm this. Eye blinks co-occur not only with stressed syllables, but with bodily pikes

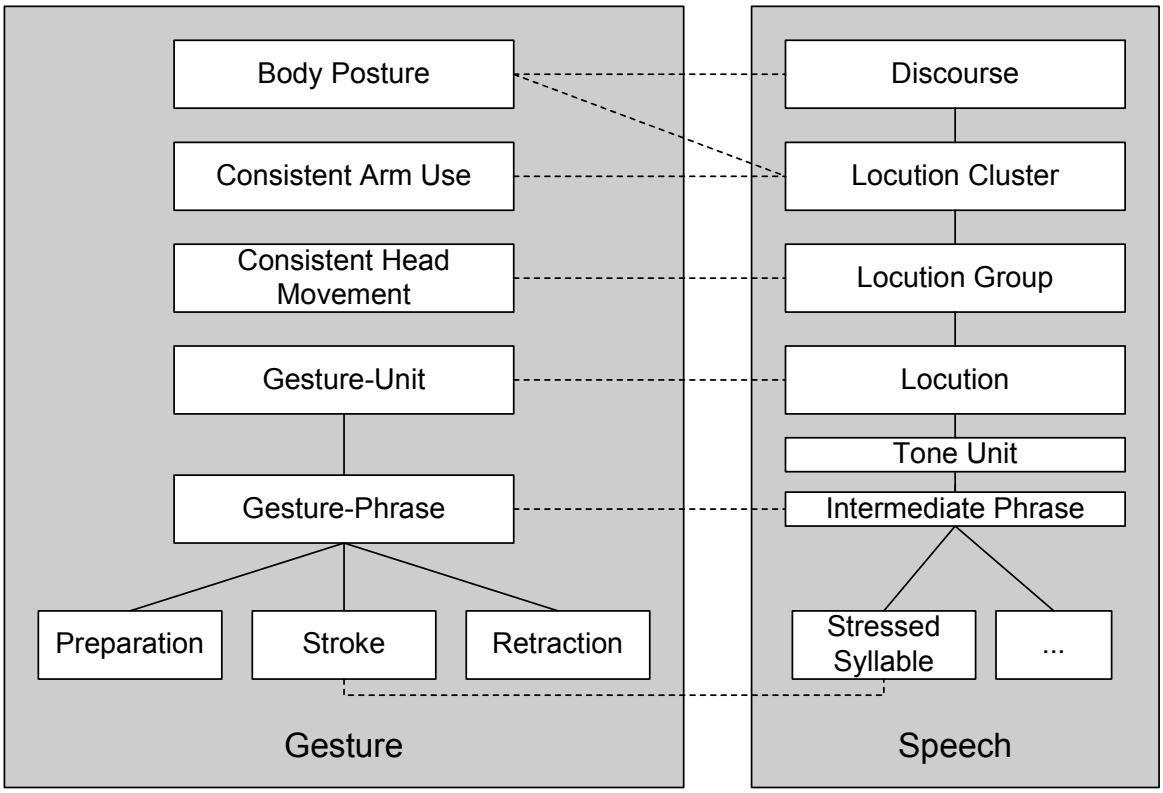


Figure 4.3: The coordination between gesture and speech.

> [Loehr's 'bodily pikes' refer to head movement apexes and gesture stroke apexes] as well. Even more intriguingly, upon close examination, eye blinks don't typically happen on the rhythmic pulse, but just prior to it, so that the eyelids are re-opening on the rhythmic pulse. It's as if eye blinks are a syncopated note, slightly anticipating the rhythmic pulse.
>
> [...]
>
> As can be seen, each eye blink is timed so that it ends (i.e. the eyes are re-opened) with other pikes (including a waveform burst), on a rhythmic pulse. This is very common in my data, and three out of four subjects timed most of their eye blinks thus. It's almost as if the speaker were holding the eyes closed until the rhythmic moment, and then opening them, just as manual gestures hold their position, and then perform the stroke at the appropriate moment. In terms of manual gestures, then, the closing of the eyelids would be the preparation, the period of closure would be the hold, and the re-opening would be the stroke. Its interesting that most eye blinks in my data took longer than the minimum apparently needed to moisten the eye. The minimum eye blink in my data lasted three frames (100 ms), yet the average was six (200 ms). The extra time could be used for the hold, to wait for the appropriate moment to re-open.

So, in summary, speech and movement are highly coordinated at different levels.

This coordination is not without trouble, and some mechanisms (e.g. hold phases) allow one to repair it in ongoing utterances. The amount of synchrony that can be achieved between speech and gesture is still an open question. De Ruiter [243] argues that speech and gesture are two ballistic processes that have few alignment possibilities. Furuyama et al. [82] however claim that the alignment of speech and gesture is interactive throughout their entire performance. The motor plan representation and execution in Elckerlyc allows for the latter, so that gesture/speech synchronization predicted by both accounts can be achieved (and be compared in synthesis).

4.4 Specifying and Executing The Motor Plan

In this chapter, I have introduced PlanUnits as elements of the motor control plan. To allow flexible coordination between PlanUnits and to allow them to interact with the world, they were designed to allow continuous timing and shape updates. I have shown that tight coordination exists between 'PlanUnits' of human motor movement, not only through mechanical connections between them, but also through neurological coupling processes.

The motor control plan consists of a coordinated combination of PlanUnits. This coordination of PlanUnits is defined by the time relations between relevant keys in PlanUnits. Such time relations can, for example, specify that a key in one PlanUnit must occur at the same time as a key in another, or that one key must occur after another. The Behavior Markup Language (BML), discussed in Chapter 6 provides the means for the specification of PlanUnits and the time constraints between them.

The next Chapter illustrates that the coordination of PlanUnits is not limited to one's own body; tight coordination is observed between the 'PlanUnits' of interacting humans. I show that the synchronization of modalities that is observed within a single person is very similar to the synchronization observed between modalities of two (or more) interacting persons and that similar models (e.g. the HKB model and its extensions discussed in Section 4.3.1) have been used to describe this synchronization. In Chapter 6, I propose an extension of BML that allows the specification of such interpersonal coordination. Chapter 7 deals with the construction and maintenance of a flexible motor plan; its execution is described in Chapter 8.

Chapter 5

Continuous Multimodal Interaction

Traditionally, interaction with virtual humans was designed using 'sender-receiver'[1] interaction paradigms, in which the user and the virtual human take turns to send (encode) and receive (decode) meaning carrying messages that travel across channels between them [151]. Such an interaction model is insufficient to capture the richness of human-human interaction (including conversation): interactions between humans are characterized by continuous interpersonal coordination. Kopp [151] classifies this coordination in:

1. *Behavior coordination*: which lets interactants assimilate their behaviors in form, content or timing;
2. *Belief coordination*: which leads to compatible knowledge about specific topics, tasks or each other;
3. *Attitude coordination*: which regulates the individual's stance toward each other or external objects.

This thesis deals with behavior coordination (here called interpersonal coordination), and specifically with the coordination of form ('shape' in this thesis) and timing. In terms of the PlanUnits defined in Chapter 4, content deals with the selection of a PlanUnit, shape deals with the parameter value selection and change within a PlanUnit, and timing deals with the placement of its keys. This chapter presents a literature overview on interpersonal coordination in interactions between humans, shows why it is useful to model interpersonal coordination for virtual humans and motivates the design of Elckerlyc and its behavior specification language BML^{T}. Some forms of interpersonal coordination have already been implemented in existing virtual human applications or frameworks. This chapter gives a brief overview of them, focusing on their architecture. The SAIBA framework provides an architecture setup for a fully functional virtual human with different layers of abstraction. I discuss how the SAIBA framework fits into virtual human applications that allow continuous interaction and which additional requirements are posed upon its behavior specification language BML to allow it to specify behavior that can be used in such continuous interaction applications.

[1] or walkie-talkie, ping-pong, vending machine [282]

5.1 Interpersonal Coordination

Interpersonal coordination [28] (coordinating our motor behavior with others) plays a large role in our social lives. Bernieri et al. [28] define interpersonal coordination as the degree to which behaviors in an interaction are non random, patterned or synchronized in both timing and shape. They categorize interpersonal coordination in *behavior matching* or similarity and *interactional synchrony*. Behavior matching includes mimicry such as interlocutors adapting similar poses. Interactional synchrony includes alignment of movement rhythm (staccato-like vs slow and fluid), synchronization of behavior (down beats at which two people change movements simultaneously) and smooth meshing/intertwining of behavior (for example smooth turn-taking and listener responses in conversation).

5.1.1 Behavior Matching

Behavior matching involves the alignment of the shape and content of the behavior of interlocutors. One important form of behavior matching is unconscious mimicry. Individuals mimic many different aspects of their interaction partners, including their postures, facial expressions, rate of speech and syntax of speech [57, 164].

Bavelas et al. [22] show how motor mimicry (in their case, responding with a winced facial expression to a person in pain) is not just a simple reflex, but forms a communicative act. The shape and timing of this facial expression is affected by whether there is eye contact with that person. If there is eye contact, an initial wince increases in intensity. If not, an initial wince might appear, but it quickly fades out.

Boker et al. [36] show that shape alignment can occur over different modalities: they attenuated the facial expressions of one of the interlocutors in a video-conferencing setup. This attenuation of facial expressions led to increased velocity of the head nods in his interlocutor.[2]

5.1.2 Interactional Synchrony

According to Clark [59], joint actions (such as conversation) can be coordinated because they divide into phases. Each phase has a unified function and identifiable entry and exit times. For example, in a handshake phases include extending the hands, shake, and withdraw. Phases can be hierarchical (the shake can be subdivided again into grasping, pumping and releasing). Synchrony requires the coordination of entry and exit times of each phase. This requires that participants can *anticipate* and *project* these entry/exit times. This entails making moment by moment timing and other (e.g. where will the ball drop in a game of catch, what will my interlocutor be pointing at) estimates.

Schmidt et al. [255] are interested in figuring out whether certain entrainment phenomena found in within-person coordination also hold for between-persons co-

[2] In another experiment they attenuated head movement, which (perhaps not surprisingly) led attenuated head movement in the interlocutor.

ordination, and whether the same very general dynamical principles govern both. They perform an experiment in which two seated participants were asked to synchronize their leg movement to be in-phase or in anti-phase while oscillating them at a tempo given by a metronome. At higher frequencies, the stability of the anti-phase pattern decreased, and involuntary transitions to the in-phase pattern occurred. Schmidt et al. demonstrate that the HKB-model, used among other things to describe *intra*-personal synchronization (see Chapter 4.3.1, [134]), can also describe the inter-personal coordination in these experiments.

In many everyday interactions, the synchronization between interlocutors is not consciously achieved. Schmidt et al. [256] provide an overview of laboratory experiments on such (unconscious) inter-personal synchronization processes. A first experiment showed that when swinging pendulums in a comfortable tempo, two subjects that can see each other unintentionally align their swinging to achieve either in-phase or anti-phase (that is, relative phase is 180 degrees) movement. The coordination was not one of absolute phase locking, but a 'non-steady state coordination behavior produced by dynamical systems with weak attractor basins and intrinsic noise'. Later experiments describe increasingly natural conditions. For example, coordinated rocking movement was observed between two participants sitting in a rocking chair and entrainment of postural sway occurs when participants interact verbally with each other in a puzzle task, even if they do not see each other. Schmidt et al. argue that the HKB-model can also describe the inter-personal coordination in these experiments.

In Chapter 4.3.2, I illustrate how the movement of our body is rhythmically organized with our speech. In dialogue, a similar rhythmic organization also occurs between interlocutors: the flow of the movements of the listener becomes rhythmically coordinated with the movement and speech of the speaker and vice-versa [60, 138]. When such interactional synchrony occurs, 'boundary' points of both speech and movement of a speaker become aligned with boundary points in the movement of a listener. In dialogue synchrony can occur on the phonic, syllable and word level of speech. In body movement the boundaries are defined by an initiation, a termination or a change in the direction of the motion in certain body parts. The listener is not just mimicking/mirroring; he aligns movement of various body parts to the speech or movement patterns of the speaker. Such alignment does not only occur with head nods, posture shift or gestures but also 'pours' into actions that are not related to the conversation (in one example in [138], the listener leans over, tamps ash off of a cigarette into an ash tray and leans back again, exactly in the rhythm of the speaker's speech). The precision of this synchrony indicates that the listener is in some way able to *anticipate* what the speaker is going to say.[3]

Furuyama [81] provides some interesting examples of synchronization between the gestures of a listener and the speech and gesture of a speaker. In an origami learning task (without paper) a learner synchronizes his 'origami-construction' gestures tightly with the speech and gestures of a teacher. The synchronization follows similar 'rules' to those of speech-gesture synchronization within one person for, for

[3]Condon proposes that the listener uses rhythmic entrainment for this; according to Kendon the prediction mechanism is of a tracking (or 'speaking while listening') nature [85].

example, stroke-alignment (see also Chapter 4.3.2). The gestures of the listener do not mimic those of the speaker, but are creative in their own right. The listener gestures sometimes even precede those of the teacher. Similar synchronization between listener gestures with speaker speech and gesture was observed (but is far less common) in a cartoon narration task. In a telephone conversation, the head nods of a listener were observed to align precisely with the head nods of a speaker.

Cassell et al. [49] show that listeners, like speakers, perform posture shifts at the boundary of a locution cluster of the utterance of the speaker.

So, in summary, the very same synchronization of modalities that is observed within a single person is also observed between modalities of two (or more) interacting persons. Such synchronization requires a prediction of the actions of the interlocutor and the alignment of one's own motor behavior to these predictions.

5.1.3 Turn-Taking

During a conversation, overwhelmingly one party talks at a time. Speaker turns are not preallocated. Interactants 'locally manage', that is on a turn by turn basis, who will be the next speaker [59, 249].

Humans are capable of very rapid turn-taking in conversation. Typically, one interaction participant starts speaking immediately after (or even before) the previous speaker finishes his turn [84, 249]. A turn switch requires the speaker to stop speaking at the right moment and the listener to take the turn immediately after this moment, producing an utterance that is relevant to both the conversation at hand in general and specifically to the utterance uttered by the previous speaker. A combination of several mechanisms has been proposed that allows humans to achieve this.

Sacks et al. [249] propose a model for turn-taking in conversation (the SSJ turn-taking model). In their model, speech is produced in turn-constructional components (TCCs, in English roughly corresponding with a sentence, clause, phrase, lexical construction). The first possible completion of the TCC constitutes a transition relevant place (TRP). The TCCs are produced in such a way that their endings are *projected* by the speaker during their execution.

Listeners can *predict* when TRPs will occur and may use their predictions to take the turn instantly at TRPs. De Ruiter et al. [244] demonstrate that the syntax of an utterance is a necessary (and possibly sufficient) cue for the prediction of TRP and shows that the intonational contour of a TRP is neither necessary nor sufficient for human TRP prediction. Barkhuysen et al. [16] show that subjects achieve better end-of-utterance classification when presented a combination of verbal and nonverbal cues instead of verbal only or nonverbal only cues. Recent work [110] has empirically shown that the timing of turn-taking is not as precise as suggested by the SSJ model. Heldner and Edlund show that in their corpora the overlap time between turns widely varies. 41-45% of all turn shifts they observed occur *after* a minimal perceivable pause (200ms). These turn shifts did not require the listener to predict the turn's end. The other turn shifts have overlapping speech or occur with

a non-perceivable pause. This suggests that some prediction mechanism is at work for these turn shifts.

5.1.3.1 Negotiating the Turn

Schegloff [251] introduces an 'overlap resolution device' as a mechanism to handle simultaneous talk by multiple participants in a conversation within the SSJ turn-taking model described above. In several cases such speech overlap is not problematic: if prior speaker is about to finish his turn, if the overlap consists of a continuer (e.g. uh huh, mm hm, yes), if the speaker allows conditional access to the turn (e.g. if he is searching for a word), or if the speech is 'choral' in nature (e.g. laughter, collective greeting). If the overlapping talk is problematic, all but one of the conversation partners should stop speaking. To display that the overlapping talk was the ground for stopping, they should do so before the end of their TCC. Schegloff lists several shape and timing adjustments of ongoing speech that are employed to keep the turn:

- Stretch the uttered sound until a TRP of the overlapping speaker, then try to say your sentence again.
- Increase volume and pitch of ongoing speech.
- Increase speech rate (when predicting an interruption by the new speaker, as if to allow no room for a new speaker to begin).
- Decrease speech rate (this is typically used when already within overlap).
- Re-utter the turn so far.
- Completely ignore the interruption and continue to speak in 'solo' mode.

These adjustments can also be used based on *predicted* (on the basis of gesture deployment, posture alignment, audible drawing of breath, or other preturn beginning behavior) interruptions. The adjustments are employed at beat (roughly syllable) granularity. When the beats of two speakers overlap, one or both of the speakers can shift to a competitive mode for the next beat (by using one of the mechanisms described above). Once the turn is secured, speech is restored to normal. Speakers are able to interrupt their speech *within* a beat.

5.1.3.2 Opportunistic Planning

The SSJ turn-taking model explains how a listener is capable to take the turn immediately after a speaker releases it. However, it does not explain how we are able to produce meaningful sentences on the spot, a problem called *opportunistic planning* by Garrod and Pickering [84].

Conversation is a joint activity in which the participants have a common goal [59]. This helps in solving the opportunistic planning problem, because the contributions of the speaker are more predictable [84].

A common mechanism to work around the need for opportunistic planning is the use of *apositional beginnings*, such as “well..” , “but ..”, “and ..”, “oh god..”, etcetera [59, 249]. Such an apositional beginning allows an interlocutor to take the turn without having a plan at hand.

Routines form another mechanism that allows humans to rapidly produce speech. A routine is an expression that is fixed to a great extent. Often it has a flat intonation. Examples are: “How do you do”, “thank you very much”, “spill the beans”, etcetera. Routines occur frequently in dialogue [219]. Routines are in general easier to produce than non-routines. The flat intonation of routines suggests that no choices are made on stress placement. Extreme examples of the use of routines are heard in the speech of radio horse racing commenters and auctioneers. These speakers have to produce rapid and time-locked monologue. They achieve this by using highly routinized language and expressions with empty slots that have to be filled (e.g. X is in the lead) [161].

Garrod and Pickering [84] argue that listening primes certain linguistic representations. Because the same representations are used in producing and understanding, these representations are activated once the listener starts to speak and he will have a tendency to use them. This process causes the internal representations of interlocutors to be aligned. This alignment applies at all linguistic levels (choice of words, sounds, grammatical forms, meanings, etc.). Interactive alignment leads to the use of routine or semi-fixed expressions by the interlocutors. According to Garrod and Pickering, such ‘dialogue routines’ greatly simplify language production and comprehension by short-circuiting the decision making processes.

5.1.4 Listener Responses

Listener responses [80] are short utterances (for example: yeah, mhm, uhu), vocalizations and/or (facial) gestures which are interjected into the speaker’s account without causing an interruption, or being perceived as competitive of the turn. Such feedback is mostly expressed simultaneously by vocal/verbal and gestural means [7]. The occurrence of listener responses has been modeled in turn-taking models by hypothesizing that they occur on a different channel than the utterance of a speaker and thus do not interfere with his turn [59]. Yngve calls this channel the backchannel [318].

Bavelas et al. [23] divide listener responses into generic responses and specific responses. Generic responses include nodding and vocalizations such as “mhm”. Specific responses such as wincing or exclaiming are tightly connected to the content of the speech of a speaker.

Specific responses require interpretation of the speaker’s utterance and generating them is cognitively more demanding than generating generic responses [23]. Jonsdottir et al. [128] suggest that humans can generate appropriate generic feedback without attending to the content of the speech. Acoustic, prosodic and lexical cues can be employed to detect a relevant position to give feedback [99, 301].

The timing of listener responses is often modulated by mutual gaze [24]. Listeners typically look more at speakers than vice-versa. If a speaker seeks a listener

response (e.g. a conformation of understanding), he looks at the listener, creating a brief period of mutual gaze. The listener is likely to respond (for example with an uhhuh, nod, etc). after which the speaker looks away. The speaker's gaze is often accompanied by pauses, changing intonation contours (e.g. rising pitch), gestures or facial displays (e.g. rising eyebrows). Quoting Bavelas et al. [24]:

> ...the timing of the listener response, is collaborative process accomplished by joint action: Speaker gaze creates the opportunity for a listener response, and the response then terminates that gaze.

In Goodwin's [95] observations, a speaker does not change the content of what he says based on the responses from the listener. Rather, the *timing* of his speech is influenced by the listener's responses. Listener responses are frequently found in complete overlap but also occur in partial overlap and silence. Goodwin states that the overlap strategy employed by the speaker depends on whether the listener feedback was a continuer or an assessment. Continuers are generic responses that simply acknowledge the receipt of the talk just heard and signal the speaker to continue speaking. Assessments are specific responses in which the listener produces an action that is responsive to the particulars of the talk. Such responses require an analysis of the content of the speaker's talk by the listener. If the speaker recognizes an assessment and is about to start a new unit, he delays this unit (e.g. by an inhalation or production of a filler) until the listener has completed his assessment. However, the speaker may deal with continuers by resuming speech before the listener response is actually finished, in effect letting continuers occur in partial overlap with the speech resumption. The importance of this is suggested by Goodwin as follows:

> ... moving to a new turn-constructional unit while the recipient's "uh-huh" is still in progress is a proper and appropriate thing for a speaker to do. Indeed this is perhaps the clearest structural way for a speaker to demonstrate that recipient's action has been understood precisely as a continuer, and to act upon that understanding.

5.2 Why use Continuous Interaction in Virtual Humans?

People tend to respond to computers and other media as they do to people. They behave as if these were social actors [227]. Thus, it is likely that an interaction with a virtual human that employs continuous interaction is more pleasant and effective for humans than an interaction with one that uses a turn-based interaction paradigm, since their (social) expectations about these virtual humans are met by the former. This section gives a short literature overview of some of the social effects of interactional coordination in human-human interaction and discusses whether and to what degree these social effects were also observed in human-virtual human interaction.

Throughout the section, I show that the exact execution (e.g. the timing, the shape and the amount) of interaction coordination can influence the perceived personality and emotional state of a (virtual) human in subtle, context dependent ways. To achieve natural interaction, one should therefore not aim for a virtual human that exhibits as much coordination with its interaction partner as possible, but rather for coordinative behavior that matches the virtual human's personality, its emotional state and the current interaction context. This thesis deals with providing mechanisms that allow the exact specification of the timing and shape of coordinative behavior and with the execution of such behavior. The *selection* of appropriate coordinative behavior on the basis of the virtual human's personality, emotional state, etcetera is beyond its scope.

5.2.1 Behavior Matching

Chartrand and Bargh [57] show that mimicking confederates are liked more than those that are not mimicking, and that interactions with mimicking confederates were rated as being more smooth. Bailenson and Yee [14] show that these effects generalize to interaction with virtual humans. A virtual human that mimics head movement of a listener (at a 4 second delay) is more liked and more persuasive than one that uses prerecorded head movement. This is an unconscious effect, the listeners were not aware that the virtual human was mimicking their movement.

Branigan et al. [39] provide a literature survey on linguistic alignment (at different levels) between people and computers. They show that people do align to computers in a similar way as they align to other people. The alignment is even stronger with computers. The authors argue that people communicating with computers use 'extra' alignment because it is unknown how well the computer will understand them. The same strategy is used when talking to non-native speakers.

Lakin et al. [164] provide a review on the social effects of unconscious mimicry. Not only does mimicry affect rapport, but this relation works in the other direction as well: rapport and interpersonal closeness can cause a person to mimic more. People mimic more when situational factors activate a desire to affiliate. The amount of mimicry is further modulated by a number of personality aspects, including empathy and self-monitoring (sensitivity to factors in the environment that may be useful; for example awareness of differences in power with the interlocutor).

5.2.2 Synchrony

When human observers perceive movement synchrony in a group of humans, they perceive this group as having rapport and being part of the same social unit [27, 163]. The degree to which individuals are perceived as a social unit is called entitativity.

The attribution of high entitativity to humans moving in (near) synchrony generalizes to observations of virtual humans, even if the type of the movement of individuals in the group is different (but still in phase) [162]. Miles et al. [194] show that the amount of attributed rapport depends not only on synchrony itself, but also

on its phase: the attributed rapport of two virtual humans (stick figures) that walk in synchrony is highest when their strides are either in phase or in anti-phase.

One of the interactional synchronization experiments discussed by Schmidt et al. [256] shows that two subjects with homogeneous social competence have less synchronous movement on a pendulum swinging task than two subjects with mixed social competence. The author explains this by noting that the measure of social competence used is correlated with social control (or dominance). The individual with the high social competence leads the individual with low social competence in the mixed pairs.

5.2.3 Turn Taking

Interruption of a speaker's turn has been associated with the display of power assertiveness, but also with the display of active and continued listening [94, 239]. Speakers that are interrupted are perceived as less assertive, more traditional and more emotionally vulnerable [239].

Ter Maat et al. show that the turn-taking strategy (e.g. the length of the pause between turns/overlap between turns) employed by a virtual human influences its perceived agreeableness, assertiveness, conversational skill and rapport [181].

5.2.4 Listener Responses

In task oriented dialog, listener responses increase the encoding efficiency. That is: fewer words are required to transmit a task-defined unit of information [160]. Kraus et al. argue that when listener responses are not available, the speaker does not have any assurance of the understanding of the listener and is therefore less likely to shorten her task descriptions.

Listener behavior has also been shown to influence the quality of a narrative of a speaker [23]. If the listener is distracted, (by making the listener count the number of days from now until Christmas) from a story told by a speaker, this reduces the number of responses, especially specific responses. Narrators telling their close-call stories to distracted listeners told them less well (they circled around, retold the ending more than once, ended abruptly, added unnecessary explanations, etc.), especially the dramatic endings.

Several virtual human systems have employed automatic analysis of surface features of speech (such as prosody) and gesture to generate generic feedback. Cassell and Thórisson show that the use of generic feedback increases the perceived language understanding and lifelikeness of a virtual human [52]. Gratch et al. [97] show that a virtual listener that uses both generic feedback and mimicry enhances the fluency of the speech of a speaker and the speaker's overall impression of the communication.

5.3 Continuous Interaction Architectures for Virtual Humans

Several virtual human applications have provided conversational interaction capabilities that go beyond a purely turn-based interaction paradigm. Here I give a brief overview, highlighting the design implications of introducing such a more continuous interaction paradigm.

5.3.1 Ymir

Ymir [283] provides an integrated framework for a virtual human, that covers both its multimodal behavior perception/interpretation and its multimodal generation on abstraction levels ranging from dialogue planning to motor behavior. It models multimodal interaction using a layered feedback-loop model (see Figure 5.1). Each of

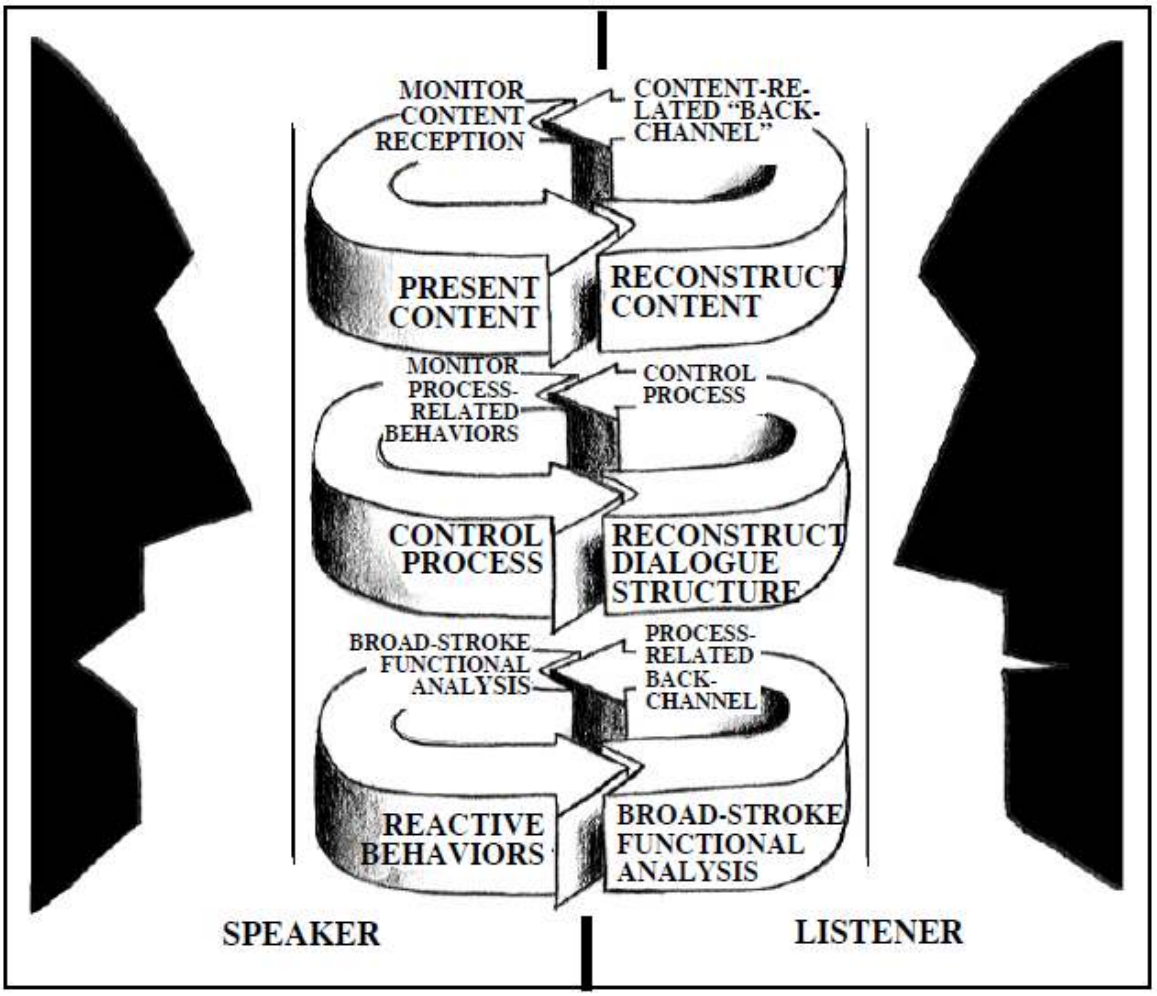


Figure 5.1: Ymir's layered feedback model, figure from [283].

these layers acts on a different update frequency and level of awareness. The Reactive Layer makes use of relatively shallow (and quick) input processing to generate

reactive behaviors (for example: look where the interlocutor is pointing, generate a generic listener feedback). The Process Control Layer deals with global aspects of dialog, dealing with issues such as when a question should be answered, what to do when information is missing, and so on. The Content Layer makes sense of the content of input and generates acceptable behavior on its basis, for example specific listener responses (e.g. indicating agreement) or executing a command issued by the interlocutor (e.g. move the blue box there).

Each layer makes semi-independent decisions, and an action scheduler arbitrates between them. Its arbitration process prioritizes reactive behavior over deliberative behavior. The action scheduler is also responsible for scheduling and executing the virtual human's behavior.

The action scheduler can execute a stream of behavior incrementally. The behaviors in this stream need to be interruptible and one should be able to change the stream to allow for relevant new events to influence it in a natural way. To achieve this, it is essential that the action scheduler keeps track of which behaviors are currently being planned, and which behaviors are executing.

5.3.2 The Listener Feedback Architecture in Max

Kopp et al. [153] propose a layered feedback-loop model that is specifically designed for the generation of listener feedback in their virtual human Max. New in this work is the use of specific feedback, on the basis of interpreted input. The feedback system in Max is part of a larger dialog management system; it is automatically activated whenever the virtual human is in a listening state. It uses a two-layered architecture.

Its planning layer consists of dedicated processes that are running to keep track of the contact, perception, and understanding listener states of the virtual human and, based on this information, decide which feedback behavior to generate and when. A reactive layer provides direct connections from input to output. These connections allow for incorporating feedback behaviors that function independently of awareness and intentional control of the sender, for example blushing, as well as behaviors that are only potentially amenable to awareness and control, such as smiles or emotional prosody.

The system uses an incremental interpretation of the input of the interlocutor. This input incrementally results in knowledge on different levels that is achieved at different time rates. These levels include perception, understanding, acceptance and emotion/attitude. In the planning layer, feedback resulting from these different levels of knowledge is selected such that feedback from the 'higher' knowledge levels is given priority. Since low level knowledge is faster than high level knowledge, this results in interesting ways to resolve conflicts between the knowledge levels:

> Max would at first look certain and nod due to a positive perception evaluation, but would then start to look confused once a negative understanding evaluation barged in, eventually leading to a corresponding verbal request for repetition or elaboration such as "Pardon me?".

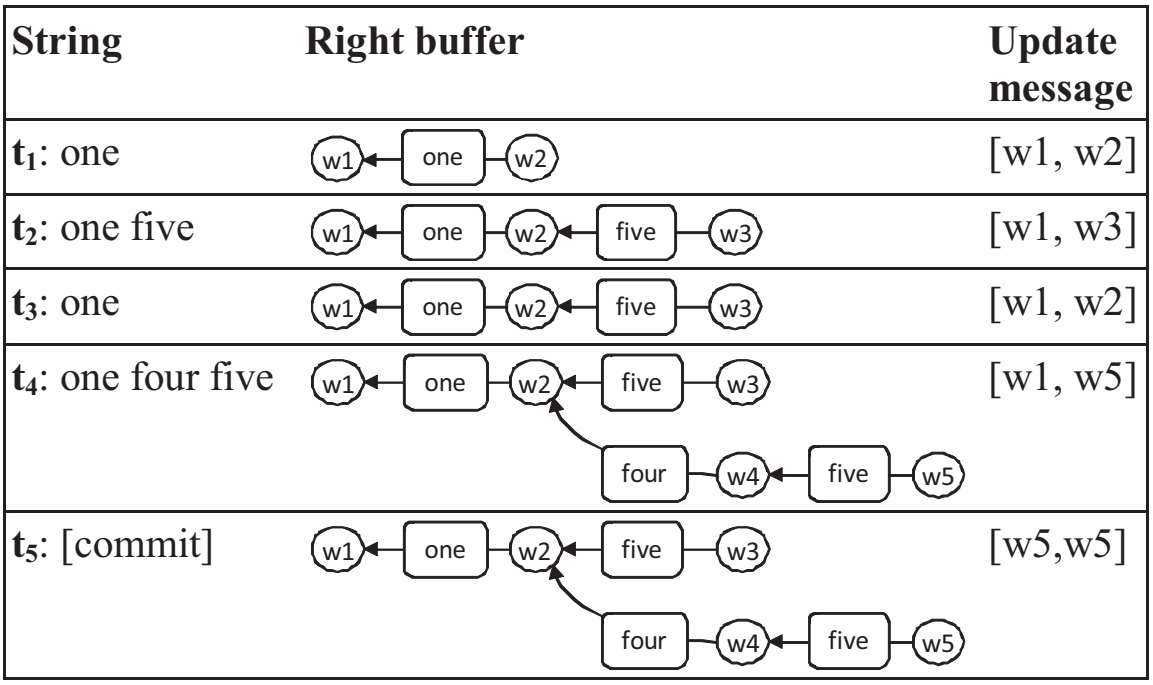


Figure 5.2: Incremental speech recognition in Jindigo, Figure from [271].

5.3.3 Jindigo

Skantze and Hjalmarsson [271] designed Jindigo, an incremental speech recognition and generation architecture that facilitates opportunistic speech planning (see Section 5.1.3.2). Jindigo is based upon a conceptual framework for incremental processing in dialog systems, proposed earlier by Schlangen and Skantze [253]. This conceptual framework describes an incremental dialogue processing system as a network of connected processing modules, where information (packaged in incremental units, or IUs) is passed along the connections. Each module has a Left Buffer, a Processor and a Right Buffer. The Processor consumes IUs from its Left Buffer and posts IUs in its Right Buffer. Each IU corresponds with a certain hypothesis. By using modules in parallel, the dialog system can manage several alternative hypotheses at the same time. Modules can be connected to each other by connecting the Right Buffer of one module to the Left Buffer of another module.

Each Processor has three basic operations. The *update* operation integrates new Left Buffer IUs into the module's internal state and eventually produces Right Buffer IUs. The *purge* operation purges a Left Buffer IUs from the internal state of the Processor. All later hypotheses build on this Left Buffer IU (represented in the internal state of other modules) must be purged as well. The *commit* operation commits to a certain hypothesis, marking it as unchangeable. For example, an input parser may commit all IUs of a sentence if its structure is complete, or a BML Realizer may commit a IU if it is completely executed.

Jindigo makes use of incremental speech detection and adapts its generation with each increment. It implements a graph-like structure to keep track of the current hypothesis of the recognized speech (Figure 5.2). Prune and purge operations

are efficiently implemented by amending to the graph without actually removing edges or vertices, even if revision occurs. At each time step, a new update message is sent to the modules that require the current/last committed hypothesis. This update message contains a pair of pointers [C, A]: (C) the vertex from which the last committed hypothesis can be constructed, and (A) the vertex from which the current hypothesis can be constructed. Speech generation in Jindigo uses SpeechPlans, which are directed graphs of SpeechUnits. The SpeechPlan is generated on the fly and captures multiple different realization options. A path on this graph is selected at run-time by an ActionManager.

In a dialog system that uses incremental processing, input hypotheses might be revised, which could lead to revisions in the ongoing SpeechPlan. To allow this, the Jindigo supports self-repairs. These repairs may be covert (they are achieved by changing planned behavior without the interlocutor noticing the plan change) or overt (involving an explicit correction). Overt revisions may include an apositional beginning (e.g. sorry, that's wrong, etc.), for example if the ActionManager decides that a correction is in order, but it does not have a plan at hand for this correction yet.

5.4 The SAIBA Framework

Over 20 years of research on virtual humans has generated increasingly sophisticated models, directed to different aspects of their behavior (using emotional modeling, computer animation, speech synthesis, etc.). Building a state-of-the-art virtual human entails re-implementing all these models. Research groups have now realized that 'the scope of building a complete virtual human is too vast for any one research group' [141].

The SAIBA initiative [152, 298] is motivated by the need to enable collaboration in building communicative virtual humans. It provides, among other things, a view on the architectural issues of building a fully functional virtual human with different layers of abstraction. It proposes a modular 'planning pipeline'[4] for real-time multimodal motor behavior of virtual humans, with a standardized interface (using representation languages) between the modules in the pipeline.

SAIBA proposes two modular splits, well explained by Thórisson and Vilhjálmsson [286]:

> The first [split] is between a representation language that describes an action/set of actions and the engine/mechanism that realizes these, according to a specification written in this language. Another split or set of splits proposed by SAIBA is between lowest-level behaviors ('animation level'), a medium-level representation typically called 'behavior' level, and a higher level called the 'functional' level. These levels correspond roughly to what have sometimes been called the primitive/servo level, e-move level and task level, respectively, in the robotics community[112].

[4] including feedback loops

[...]

> The idea behind the split between representation language and realization engine is to enable those researchers who desire to focus on a particular level of planning to stick to a certain level of detail. The language describing the desired outcome at a particular detail level can be represented in a common way between researchers, making easier the collaboration on and competition between proposed mechanisms. This allows construction of alternative planning mechanisms at particular levels of abstraction, and thus exploration of different ways of producing certain behavior phenomena, without having to solve the mechanism for the whole field, as the representational languages provide an API that allows modular sharing of solutions for different parts of the architecture. This also enables the comparison between realization mechanisms from different research labs.

The SAIBA Intent Planner module generates a plan representation on the functional level, specified in the Functional Markup Language (FML). The SAIBA Behavior Planner generates a plan representation that is incrementally specified through blocks written in the Behavior Markup Language (BML) (see Figure 5.3).

Figure 5.3: The SAIBA architecture.

A BML block (see Figure 5.4 for a short example) describes the occurrence of certain types of behavior (facial expressions, gestures, speech, and other types) as well as their relative timing. This relative timing is described by constraints between synchronization points (see Figure 5.5 for the standard synchronization points in each behavior).

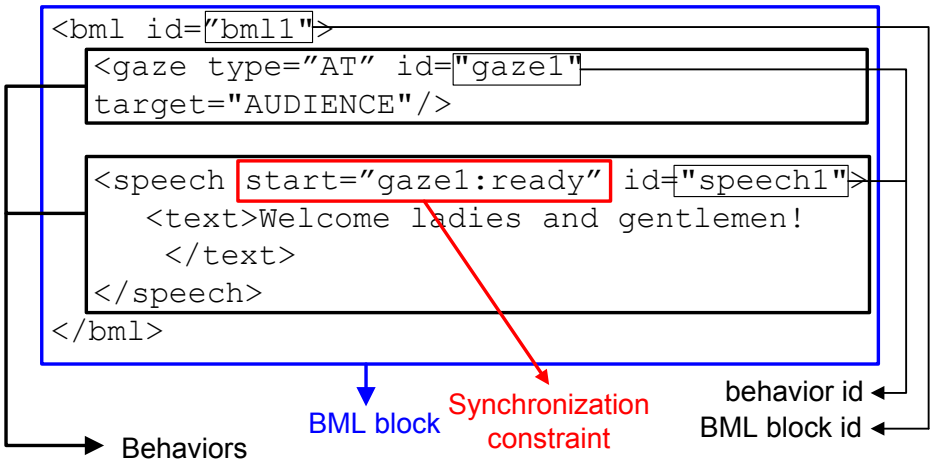


Figure 5.4: An example of a BML request containing a gaze and a speech behavior. A synchronization constraint ensures that the speech starts as soon as the gaze is aimed at the audience.

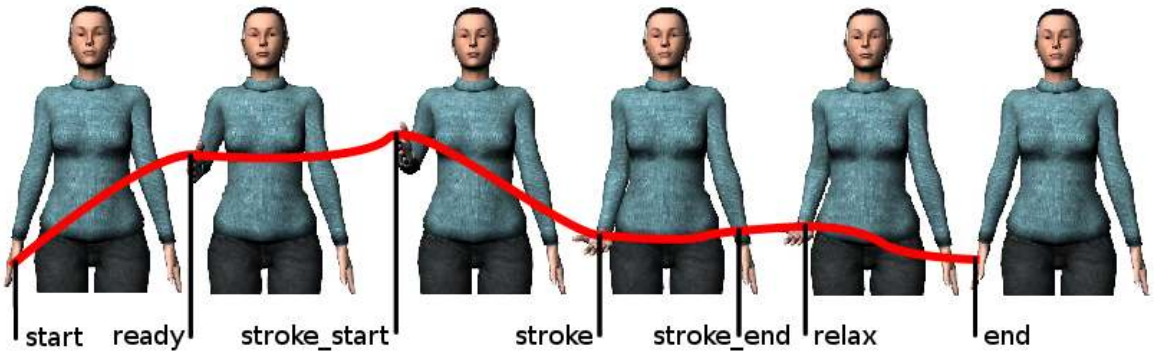


Figure 5.5: Standard BML synchronization points

FML will represent what an virtual human wants to achieve: its intentions, goals and plans [113]. The exact syntactical representation for this is still under discussion. Heylen et al. [113] indicate that (among other things) context, communicative actions, content, mental state and social-relational goals could be elements in FML.

5.5 Continuous Interaction in the SAIBA Framework

At first glance, SAIBA's pipelined architecture seems to conflict with the layered feedback architectures proposed by other integrated virtual human frameworks (see Section 5.3.1 and Section 5.3.2). However, one does not necessarily need to interpret the SAIBA architecture as a pipeline in which behavior is only generated on the basis of intent from the SAIBA Intent Planner and then transformed by a SAIBA Behavior Planner 'function' into a behavior suitable for a Realizer. Instead, Intent Planning and Behavior Planning can be regarded as processes that run at different update frequencies and that can each generate behavior descriptions, at different

levels of abstraction. This view (illustrated in Figure 5.6) allows several feedback loops to exist in the SAIBA framework.

In the outer feedback loop (indicated with the black arrows) the SAIBA Intent Planner makes use of *interpreted* user behavior to decide on the Intent of actions that are to be executed by the virtual human.

Bevacqua et al. [29] argue for a feedback loop (indicated with the gray arrows), that generates sensor-activated unconscious and unintentional (so not originating from the Intent Planner) behavior in the SAIBA Behavior Planner. One example of such behavior is mimicry, which they propose to implement by incrementally submitting BML blocks to the Realizer.

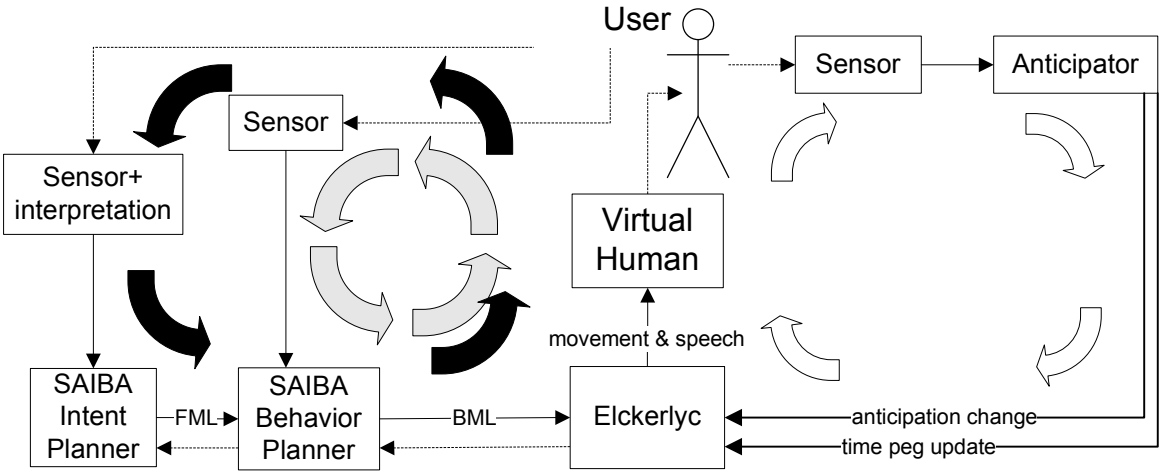


Figure 5.6: SAIBA as a layered feedback architecture.

5.5.1 A Feedback Loop at Realization Level

A novel aspect in Elckerlyc is that it contains a feedback loop at realization level which elegantly allows the specification and realization of synchronization to predicted and incrementally updated behavior events of the virtual human's interlocutor. Such a feedback loop achieves the tight synchronization needed for, for example, interactional synchrony.

In Section 5.1.2, I showed that the synchrony between behavior of different humans relies on the very same synchronization mechanisms that exist between modalities (e.g. speech, gesture) within one's own body. Therefore, it makes sense to allow the specification of synchronization to behavior of an interlocutor in the same fashion as synchronization of modalities within one's own body. The BML description of behavior would thus become a specification of a part of a joint action, rather than the specification of completely autonomous behavior (see BML Example 1).

In BML, synchronization between behaviors of a virtual human on different modalities (e.g. speech, gesture) is specified as the alignment of the synchronization points of these behaviors. The SAIBA Behavior Planner specifies only the existence of such time constraints, the exact timing of the constraints is determined in the Realizer. If synchronization to an interlocutor is to be specified in an analog fash-

ion, synchronization points that represent (predicted) synchronization points of the behavior of an interlocutor are required. The SAIBA Behavior Planner should then only be required to specify the existence of the constraints between interlocutor synchronization points and behavior synchronization points, their exact timing is determined at Realization level. I have introduced the Anticipator as generic module[5] to predict the timing of such external (e.g. interlocutor) synchronization points and provide a BML extension that can specify their alignment to the behavior of a virtual human.

BML Example 1 Specifying taking the turn as an autonomous action and as part of a joint action.

(a) Specifying a speech behavior that takes the turn as an autonomous action. The start of the turn is defined by the SAIBA Behavior Planner.

```
<bml id="bml1">
  <speech id="speech1" start="3">
    <text>Bla bla</text>
  </speech>
</bml>
```

(b) Specifying a speech behavior that takes the turn as part of a joint action. The start of the turn is determined by the (predicted) end of the turn of the interlocutor.

```
<bml id="bml1">
  <speech id="speech1" start="anticipators:speechStopAnticipator:stop+0.5">
    <text>Bla bla</text>
  </speech>
</bml>
```

An Anticipator typically predicts time events on the basis of observed interlocutor behavior. These predictions are often incremental, and increase in precision over time. Elckerlyc provides automatic adaptation of the behavior plan it manages in reaction to updated time predictions of an Anticipator. This incremental update of synchronization constraint timing introduces another feedback loop (indicated with the white arrows), which is located within the Realization process. This feedback loop allows small modifications based on user observations to be made to ongoing behaviors directly. Such adaptivity is not only useful for updating the behavior plan in reaction to an interlocutor, it is also potentially useful to allow minor timing updates to maintain consistent timing with 'internal' modalities that provide an incremental prediction of the timing of their sync points (that is, those that have a poor initial idea of what their timing will be like, for example the behavior of some robots).

[5]Implementations of these modules could include an end-of-turn Anticipator (see Chapter 12.2.2 for possible implementations), or a gaze-end Anticipator.

5.6 Discussion

In this chapter, I have illustrated the interpersonal coordination that occurs in humans and shown that it is beneficial to implement virtual humans that are capable of such interpersonal coordination. Implementing such continuous interaction has several consequences for intent planning, behavior planning and behavior realization. Most of this thesis deals with continuous interaction for behavior realization. In this section I briefly discuss the consequences of continuous interaction on intent and behavior planning and the specification requirements for behavior realization of continuous interaction, all within the SAIBA framework.

5.6.1 Consequences of Continuous Interaction for SAIBA Intent and Behavior Planning

I propose that to achieve continuous interaction, one should think of the SAIBA planning processes as processes that construct a multimodal behavior plan on the fly. This multimodal behavior plan serves as a *joint* information structure shared by the SAIBA Intent Planner, SAIBA Behavior Planner and the Realizer. At first one might think that a behavior plan is merely an implementation aspect, internal to SAIBA components such as the Behavior Realizer. This is in line with the idea that languages such as FML and BML are declarative languages, so a stream of BML behaviors would carry no explicit state information. But for certain continuous interaction mechanisms this in an untenable position. An important example of why this is the case is that of interruption of speech: a SAIBA Behavior Planner sending a speech behavior for a single sentence to the Behavior Realizer cannot prevent the environment, including humans, to interrupt that sentence at a later time. When that happens, continuous interaction demands that there is an instantaneous reaction, for instance by means of sending an interrupt behavior that partially cancels the speech behavior and removes it from the behavior plan, possibly replacing it with some alternative behavior. The important observation here is that all this implies that SAIBA Behavior Planners, and ultimately also SAIBA Intent Planners, must be aware of the state (in the form of a behavior plan or otherwise) of the Realizer. An Intent Planner, for instance, can use feedback concerning behavior progress and interrupt information to decide whether some message can be considered to have actually 'arrived' at the (human) receiver, or not, and then act accordingly. Based on similar information the dialogue management functionality inside the Intent Planner and the Behavior Planner can make inferences about who has the floor at any moment. (Although 'owning the floor' is clearly a higher level concept, it is affected nevertheless by lower level events such as interruption, gaze behaviors, etc.). Note that interruptions are considered less of a problem in more traditional turn-based dialogue systems, where the (unrealistic) assumption is made that utterances from interlocutors will alternate. In that case, an interruption would only be noticed at a higher level, say at the Intent Planning level. This has the undesirable effect that interruptions at a lower level are handled in an unnatural manner: a virtual human that is interrupted in a turn-based dialog system would simply keep talking as

if nothing has happened, unaware of the interruption until the next 'round' in the dialogue.

Continuous interaction therefore requires that planning and execution information is shared between the planning processes (see also Section 5.3.1 for a similar argument in the Ymir architecture, and Chapter 6.5.2.3 for a scenario that illustrates this in detail). In the SAIBA framework, plan sharing is achieved by implementing the feedback channels that communicate progress information between the SAIBA Planning processes.

5.6.2 Requirements for the Specification of Continuous Interaction in BML

Continuous interaction, involving (among other things) behavior matching, interpersonal behavior synchrony, rhythmic alignment, fluent turn-taking, turn overlap management and listener responses requires an interface with a BML Realizer that provides capabilities that go beyond what can be expressed in BML. BML can specify the internal (within the virtual human) synchronization constraints between behaviors (e.g. speech, gesture) and provides a static description of the form of each behavior.

In addition to this, continuous interaction requires the specification of:

1. *the synchronization of (ongoing) behavior to predicted events* originating from the environment or the virtual human's interlocutor. The timing of such events is incrementally updated.

2. *instant interruption and fluent continuation* of ongoing behavior.

3. *modifications* in the shape of ongoing behavior (e.g. speak louder).

4. *immediate execution of behavior*, allowing apparent opportunistic planning.

Table 5.1 shows what capabilities are required for different aspects of interactional coordination. The next chapter demonstrates how I have specified these capabilities in HMI's BML extension BML^{T}.

	behavior matching	synchrony of behavior	rhythmic alignment	fluent turn taking	turn overlap management	listener response
Allow the alignment of the entry/exit phases of behavior to predicted and incrementally updated interlocutor events		X	X	X	X	X[1]
Allow instant interruption and fluent continuation of ongoing behavior					X	X[2]
Allow modifications of the shape of ongoing behavior	maybe[3]	maybe[4]	maybe[4]		X	
Immediate execution of behavior, apparent opportunistic planning				X	X[5]	maybe[6]

[1] For speakers and listeners.
[2] For speakers.
[3] In current virtual human platforms that support mimicking and alignment, these are achieved incrementally using subsequent behaviors. However, some forms of mimicking (see e.g. [22]) might require shape adjustments within *ongoing* behaviors.
[4] For most behavior types, it is unlikely that the timing can be modified completely independent of its shape.
[5] In restarts.
[6] Generic listener responses could rapidly be generated using routinized speech.

Table 5.1: Specification requirements of some aspects of interpersonal coordination.

Chapter 6

On the Specification of Multimodal Continuous Behavior for Virtual Humans

The Behavior Markup Language (BML) has become the de facto standard for the specification of the synchronized motor behavior (including speech and gesture) of virtual humans. BML is interpreted by a BML Realizer, that executes the specified behavior through the virtual human it controls.

This chapter first discusses the historical roots of BML, its behavior and synchronization description features and the underlying design considerations. Continuous interaction applications with virtual humans pose several generic requirements on the specification of behavior execution, beyond that of multimodal internal (that is, within the virtual human) synchronization and form descriptions provided by BML. I outline the need for the detailed specification of interruption of ongoing behavior, the change of parameter values in ongoing behavior (e.g. speak louder) and the need for synchronization with predicted external time events (e.g. originating from the interlocutor). Several usage scenarios further illustrate these needs. BML Twente (BML^T) extends BML by providing the *specification* of the continuous interaction capabilities discussed above. It thus provides a SAIBA Behavior Planner with a generic interface to a Realizer through which continuous interaction can be realized. I conclude the chapter by discussing the consequences that the use of BML^T poses on a SAIBA Behavior Planner that constructs it and highlights the additional requirements posed upon a BML Realizer that is able to execute BML^T.

6.1 Specifying Multimodal Behavior for Virtual Humans: A Brief History

There are many languages to specify the motor behavior of virtual humans on multiple modalities. One of the key aspects in designing these specifications is the description of the (time) alignment between behaviors on the different modalities. In classic multimodal systems that generate behavior for virtual humans

[50, 53, 105, 209, 279], the speech and gesture modalities are aligned by timing the gestures to speech timing events generated by a speech synthesizer. Speech then guides the timing of the gestures; speech is the leading modality [304]. This is reflected by the behavior specification used in these systems: they typically consist of the text to be spoken annotated in-line with nonverbal behavior elements [53, 65, 209, 279] (see also Example 1).[1]

Example 1 An example of a script, taken from [279]. Note how the gestures are specified in-line with the text to be spoken.

```
you go <g type=iconic>UP</g> the <g type=deictic>STAIRS</g>
<g type=iconic>turn SHARPLY to the RIGHT</g>
and go through the <g type=beat>FIRST</g> door
```

However, speech/gesture timing in humans shows more complex coordination. For example: speech output can be delayed so that a complex gesture can be finished or a gesture's hold phase can be used to correct the timing of a gesture that was started too early ([154], see also Chapter 4.3.2). MURML [150] (see Example 2) is the first gesture and speech synthesis specification that does not require speech as a leading modality. It defines the speech and gesture in separate channels and uses symbolic synchronization points to define their alignment. The multimodal

Example 2 An example of a MURML script, taken from [150]. Note how the speech and gestures are specified in separated channels and how the synchronization is achieved through symbolical synchronization points $t1$, $t2$, $t3$ and $t4$.

```
<definition>
    <utterance>
        <specification>
            And now take <time id="t1"/> this <time id="t2"/> bar
            <time id="t3" chunkborder="true"/> and make it <time id="t4"/> this
            big. <time id="t5"/>
        </specification>
        <focus onset="t1" end="t2" accent="H*"/>
        <behaviorspec id="gesture_1">
            <gesture id="pointing_to">
                <affiliate onset="t1" end="t3"/>
                <param name="refloc" value="$Loc-Bar_1"/>
            </gesture>
        </behaviorspec>
        <behaviorspec id="gesture_2">
        ...
        </behaviorspec>
    </utterance>
</definition>
```

[1] RRL [220] is a notable exception. While the behavior planning pipeline used in NECA implies that speech timing is enforced by gesture timing, gestures are specified in a separate channel.

behavior specified in MURML is scheduled as a concatenation of chunks, based on McNeill's [189] segmentation hypothesis. The chunks contain a segment of speech and/or one gesture, which are synchronized using symbolic synchronization points. If only arm gestures and speech are to be coordinated, such an approach works fine. However, in many multimodal generation applications the behavior of virtual humans is not confined to just gesture and speech: virtual humans could operate and explain the working of complex machinery [126], behave naturally in a war zone [236], comment on an ongoing soccer match [9], or simply walk through an environment while conversing. To support the specification of the coordination of such a wider range of modalities, I designed MultiModalSync during my master's studies [303, 304]. MultiModalSync implicitly defines a leading modality, but this leading modality can change over time (see Example 3). Such a change does not emerge from the behavior generation itself, it has to be specified beforehand.

Example 3 An example of a MultiModalSync script, taken from [303]. Note how the speech, presentation sheet changes and gestures are specified in separated channels and how their synchronization is achieved through symbolical synchronization points $t1$, $t2$ and $t3$. Channels can both set synchronization points, using DefineSynchronisationPoint and make use of synchronization points defined by other channels. This allows an explicit specification of the leading modality over time.

```
<MultimodalSync>
    <Segment>
        <Channel name="sheetcontrol">
            <UseSynchronisationPoint value="t1"/>
            <ChangeSheet name="sheet1">
            <DefineSynchronisationPoint id="t2"/>
        </Channel>
        <Channel name="verbal">
            <UseSynchronisationPoint value="t2"/>
            So, the <DefineSynchronisationPoint id="t3"/> bookshelf
            right now is sitting here.
        </Channel>
        <Channel name="deictic">
            <Point target="bookshelf" stroke="t3">
        </Channel>
    </Segment>
</MultimodalSync>
```

In the Behavior Markup Language[2] (BML) [152, 298] (see Figure 6.2), researchers working on virtual humans (including those who designed the systems and specification mentioned above), collaborate to provide a behavior specification language that allows the specification of conversational (such as speech, gesture) and other (such as locomotion and posture) behaviors of virtual humans. BML allows very flexible time synchronization mechanisms that do not require the notion of a leading modality.

[2] http://www.mindmakers.org/projects/BML

This chapter discusses the design choices the SAIBA initiative (including myself) has made in BML, and how I have extended BML to support the specification of multimodal virtual human behavior in applications that require continuous interaction.

6.2 BML

The SAIBA Framework (discussed in detail in Chapter 5.4), describes a generic architecture for virtual human applications. It contains a three-stage process: communicative intent planning, multimodal behavior planning, and behavior realization (see Figure 6.1). BML provides an interface between the SAIBA Behavior Planner and a Realizer.

Figure 6.1: The SAIBA framework.

Initially, BML was designed to provide a general, Realizer-independent description of multimodal behavior that can be used to control a virtual human. This description ranged from the mere occurrence and the relative timing of the involved behavior, to the detailed (yet Realizer-independent) definition of the behavior's form [152]. Realizer independence requires that the BML specification cannot rely on such things as the existence of certain joints in the skeleton of the virtual human, the availability of a specific speech synthesis system, the availability of animation files, and so on. Instead, BML uses body parts, lexicalized locations and common verbs (for example: speech, face, gesture, center, left) as terms in its specification.

At a later stage (see [298]), the requirement for detailed form descriptions was dropped for various reasons:

1. It is hard to unify all the different detailed form descriptions by all research groups into a single behavior description.

2. Even if this were to succeed, such a description would require a new Realizer to implement an unwieldy amount of behavior variation.

3. Many more or less standard representation languages already exist that provide detailed form descriptions for a single modality (for example: SSML, MaryTTS, or Microsoft SAPI for speech).

Therefore the SAIBA initiative opted to design a simple 'Core' BML description that describes the behavior's form in a coarse manner and to provide extension mechanisms to allow BML authors to specify the form of their behaviors in greater detail whenever desired.

A BML stream contains BML blocks with behaviors (such as speech, gesture, head movement, etc.) and specifies how these behaviors are synchronized (see also Figure 6.2). Synchronization of the behaviors to each other is done through BML constraints that link synchronization points in one behavior (start, end, stroke, etc; see also Figure 6.3) to synchronization points in another behavior. Each behavior and each BML block is identified by its id.

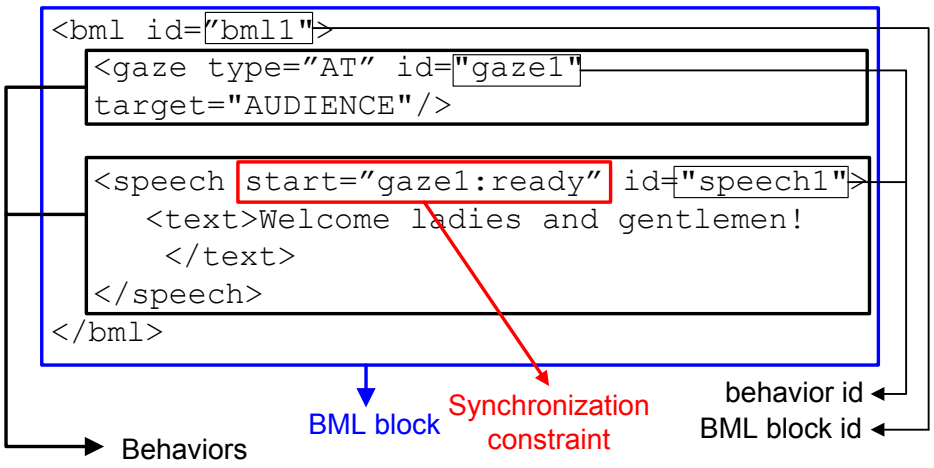


Figure 6.2: An example of a BML request containing a gaze and a speech behavior. A synchronization constraint ensures that the speech starts as soon as the gaze is aimed at the audience.

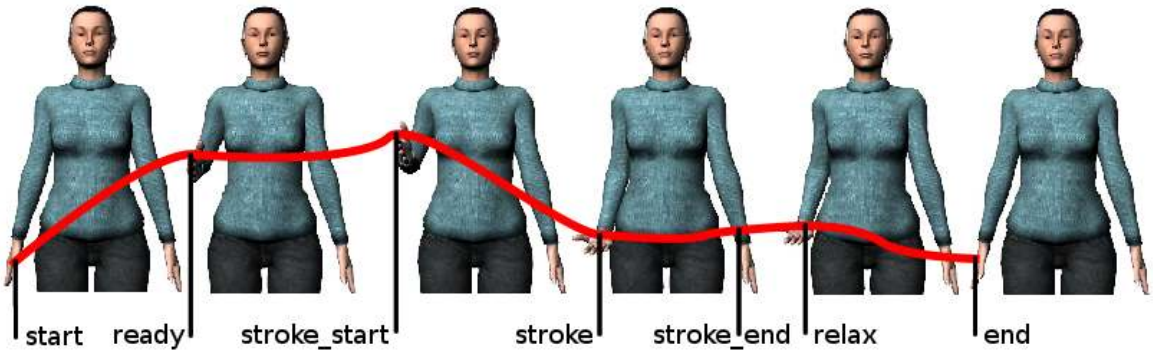


Figure 6.3: Standard BML synchronization points

The BML specification does not prescribe a semantic meaning for the BML block. The BML block merely provides a convenient mechanism to cluster and stream behaviors. This allows users of BML to specify short spurts of behavior (for example: using 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 block (for example as entire monologues).

6.2.1 Scheduling BML

A BML Realizer receives a continuous stream of BML blocks. These blocks are to be scheduled in order of their arrival. This scheduling of a sequence of BML blocks results in a multimodal behavior plan.[3] BML Scheduling thus updates the current multimodal behavior plan on the basis of a provided BML block (see Figure 6.4).

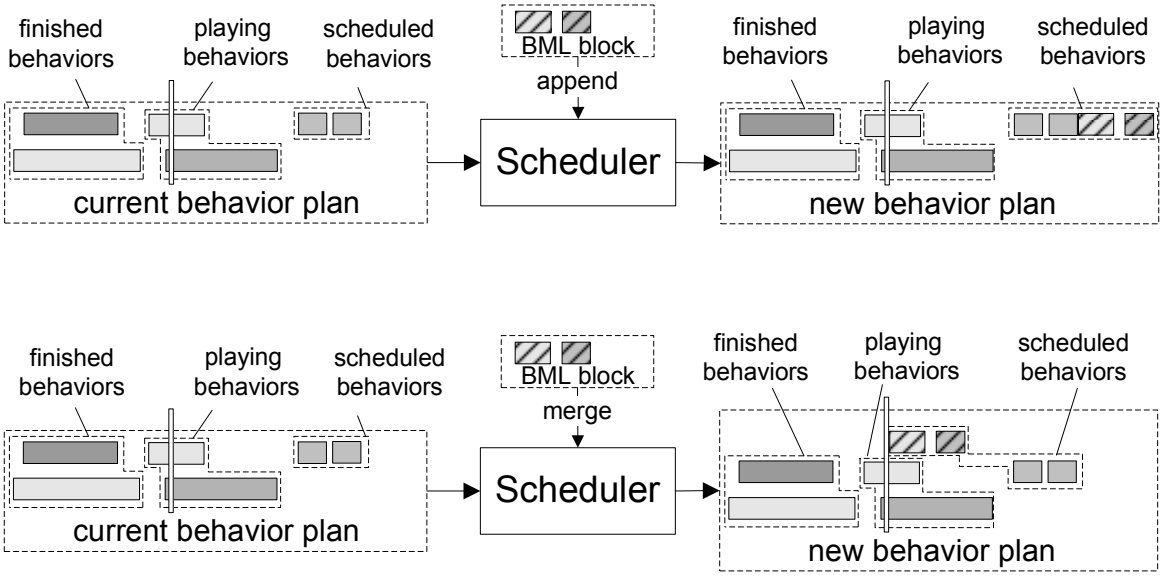


Figure 6.4: The scheduling process. The white bar indicates the current time. The new BML block defines how the currently playing and planned behaviors are updated and which new behaviors are inserted, using a scheduling attribute. `append` (top) indicates that the behaviors in the BML block are to be 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.

The SAIBA Behavior Planner can indicate how a BML block is to be combined with the current behavior plan using the `scheduling` attribute in the `bml` element. A BML Realizer should implement the following mandatory scheduling attributes[4]:

1. *replace*: completely replaces the current behavior plan by the behaviors specified in the new BML block; interrupts all running behavior.

2. *append*: schedules the behaviors specified in the new BML block to play after *all* BML blocks in the current behavior plan have finished.

3. *merge*: merges the behavior specified in the new block with the current behavior plan. That is, behaviors in the new block are added to the current plan and do not have to wait for any behaviors from the previous blocks. An exception arises when some new behavior would conflict with some behavior already present in the current plan; in that case the newer one will be ignored, and a

[3]The multimodal behavior plan is an abstract representation of the Motor Plan discussed Chapter 4. A behavior might be represented by multiple PlanUnits in the Motor Plan.

[4]see also `http://wiki.mindmakers.org/projects:bml:multipleblockissue`

notification is sent back to the SAIBA Behavior Planner. Behavior descriptions in a merge block are not allowed to refer to the timing of behaviors outside the BML block.

If no scheduling attribute is specified, the block is scheduled using the merge attribute.

6.2.2 Behaviors

Core BML supports the following set of behaviors:

- `locomotion`: move the body of the virtual human from one location to another. Can be specified by target in the world or by movement velocity for continuous movement.

- `posture`: put the body in an 'overall physical configuration', specified mainly by a stance (standing, sitting, lying, ...) and a shape (crossed, open, stretched, ...).

- `speech`: generate both speech audio (or text) and lip/mouth movement, for example using a speech synthesizer and viseme morphing. Specified by the text that is to be spoken.

- `gesture`: coordinated movement of the arms and hands. Specified mainly by a type (e.g. point, beat, conduit, lexicalized, ...). Lexicalized gestures are linked to a certain animation or controller by specifying a lexeme (for example: the filename of the animation, the id of the controller).

- `head`: head movement, to be interpreted as an offset from posture and gaze direction. Includes head shakes, head nods and generic head rotations.

- `face`: movement of the facial muscles to form certain expressions. The exact specification format is under discussion at the time of writing. It has been proposed to use Action Units from Ekman and Friesen's Facial Action Coding System [71] and/or to use lexicalized face expressions in a similar way to lexicalized gesture.

- `gaze`: Coordinated multi-joint movement, indicating where the character is looking. Specified mainly by a gaze target, and a set of gaze modalities (eyes, head, neck, torso, legs).

- `emit`, `wait`: the `wait` behavior serves both as a simple 'performance pause' behavior, typically used to delay the execution of other behaviors and as one of the elements of an event/message based synchronization system that is to be used to synchronize inter-virtual human behavior. The semantics and syntax of the latter are still under discussion at the time of writing.

6.2.3 Synchronization

Every behavior is broken down into six phases of realization, inspired by the phases of gesture (see [189]). Each phase is bounded by a sync point (See Figure 6.3). Behavior may also define custom sync points (for example using `sync` in speech, see BML Example 2). These custom sync points must always occur after the start and before the end sync point of the behavior.[5]

BML Example 2 A speech behavior that defines the custom sync point sync1 to occur directly after the word custom.

```
<speech id="speech1">
  <text>This speech behavior defines a custom <sync id="sync1"> sync point.
  </text>
</speech>
```

Synchronization constraints are used to specify the timing relation between two or more *sync references*. A sync reference consists of either an offset (in seconds) from the start of the BML block, or a triple of a behavior, a sync point and an offset time from the sync point in the behavior (in seconds).

The `constraint` element provides the general mechanism to specify time constraints between sync references. In core BML one can specify that:

1. Two or more sync references should occur at the same time (See BML Examples 3a,3c).
2. One or more sync references should occur at a fixed time after the start of the BML block (See BML Example 3b).
3. One or more sync references should occur before or after a specified sync point (See BML Example 3d).

A shorthand is provided to specify the synchronization of a standard sync point of a behavior to a sync reference within the behavior specification itself (see the synchronization constraint in Figure 6.2, BML Example 3a shows the generic way to specify the same constraint). All synchronization constraints are interpreted as bi-directional constraints. That is: the synchronization constraint in Figure 6.2 states that the start synchronization point of speech1 must occur at the same time as the ready synchronization point of gaze1, but allows that the timing of either or both behaviors is modified to satisfy the time constraint.

[5] BML does currently not define where custom sync points are positioned in relation to the other five standard sync points.

BML Example 3 Several BML blocks illustrating BML's general synchronization mechanisms.

(a) The start sync of speech1 is synchronized with the ready sync of gaze1. This bml script is equivalent to that used in Figure 6.2.

```
<bml id="bml1">
  <gaze type="AT" id="gaze1" target="AUDIENCE"/>
  <speech id="speech1">
    <text>Welcome ladies and gentlemen!</text>
  </speech>
  <constraint id="c1">
    <synchronize ref="speech1:start">
      <sync ref="gaze1:ready"/>
     </synchronize>
  </constraint>
</bml>
```

(b) sync points ready of gaze1 and start of speech1 should occur 4s after the start of the BML block.

```
<constraint id="c1">
  <synchronize ref="4">
    <sync ref="gaze1:ready"/>
    <sync ref="speech1:start"/>
  </synchronize>
</constraint>
```

(c) speech1:sync4 should occur at the same time as beat1:stroke and 0.5s after nod1:stroke.

```
<constraint id="synchronize_example">
  <synchronize ref="speech1:sync4">
    <sync ref="beat1:stroke"/>
    <sync ref="nod1:stroke+0.5"/>
  </synchronize>
</constraint>
```

(d) This constraint requires that the ready sync of gaze1 occurs before the start sync of speech1.

```
<constraint id="before_example">
  <before ref="speech1:start">
    <sync ref="gaze1:ready"/>
  </before>
</constraint>
```

6.2.4 Feedback

A Realizer should inform the SAIBA Behavior Planner of any errors that occur when realizing the BML blocks that are sent to it. If required (that is, a failure would otherwise occur), a Realizer may decide to drop some constraints, or to execute behaviors in a slightly different form than requested by the SAIBA Behavior Planner. The SAIBA Behavior Planner is informed of this using a warning message. The SAIBA Behavior Planner needs to monitor the progress of the multimodal plan it composed for various reasons. For example, it might need to know when it would be an appropriate time to merge some new behaviors, or when exactly certain information has been delivered to a user. For Realizers that support the interruption of ongoing behaviors, monitoring behavior progress is crucial to set up the appropriate interruption strategy (see also the scenario in Section 6.5.2.3).

A Realizer is expected to provide a SAIBA Behavior Planner with feedback on the current state of any BML block, and to notify it of execution warnings and exceptions. BML does not specify a specific format for feedback, exception and warning messages, but does specify the minimum content of such messages. A Realizer must at least provide the following feedback:

1. *Performance start feedback* Indicates that the Realizer has started executing the BML block. Must include the BML block id of the started block.

2. For each sync point in each behavior in the BML block: *Sync-Point Progress Feedback*. Indicates that a sync point in one of the behaviors of the BML block has passed. Must provide the sync points BML block id, behavior id and sync id.

3. *Performance stop feedback* Indicates that the Realizer has finished executing a BML block. Must provide the BML block id and the reason for stopping (for example: successful completion, error).

4. *Warning feedback* is used to notify the SAIBA Behavior Planner that requested behaviors and/or synchronization constraints were modified during realization. Warning feedback must contain the BML block id of the block in which the warning occurred and the list of behavior ids and synchronization constraints that were modified to facilitate synchronization.[6]

5. *Exception feedback* is used to notify the SAIBA Behavior Planner that requested behaviors and/or synchronization constraints have failed to be realized. The exception feedback must include the BML block id in which the exception occurred, and whether the BML block was completely canceled. If the block is not completely canceled, it must also provide the list of behavior ids and synchronization constraints that failed. If the BML block failed entirely, a reason

[6] In the current version of Elckerlyc, behaviors are dropped if constraints on them cannot be satisfied. Elckerlyc does not have mechanisms in place to drop constraints rather than behaviors, or to modify the form of a behavior to satisfy constraints. Therefore, Elckerlyc does not make use of warning feedback.

for this failure must be provided (for example: BML parsing error, non-existing gaze target, synchronization constraints could not be achieved, etc.).

6.2.5 Mechanisms to Extend BML

Allowing Realizer specific extension of BML is crucial for wide acceptance of the language. However, this conflicts with the desired Realizer-independence of the language. BML resolves this conflict by proposing a simple, small set of core BML behaviors, and allowing additional descriptions to describe these behaviors in more detail. Such additional descriptions can make use of existing unimodal behavior description languages, so that they still might be shared by different Realizers. For example, Realizers from several research groups might allow the description of speech in SSML.

The set of BML behavior elements is by no means comprehensive, as much of the ongoing work behind BML involves identifying and defining a broad and flexible library of behaviors. Implementors are encouraged to explore new behavior elements. In summary: BML extension is handled in the following ways (see also Figure 6.5):

1. Additional behaviors should be designed as new XML elements using custom XML namespaces (see BML Example 4).

2. Specialized attributes can be used to extend core BML behaviors. Such attributes should be identified as non-standard BML by utilizing XML namespaces (see BML Example 4).

3. Behavior Description Extensions provide a principled way of specifying core BML behaviors in a more detailed manner, typically employing an existing XML language (see BML Example 5).

BML Example 4 A customized animation behavior (sbm:animation) and a customized joint-speeds attribute (sbm:joint-speeds). The latter specifies the BML gaze behavior in a more detailed manner. Both extensions are from the SmartBody project[280].

```
<bml 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>
```

Behavior Descriptions Extensions go beyond the core BML behavior specification in describing the form of a behavior. They allow a BML author to describe the behavior in more detail for a Realizer that supports the extensions, while still providing the core behavior as a fallback for Realizers that do not. The additional descriptions are embedded within the element of the behavior they extend as children elements.

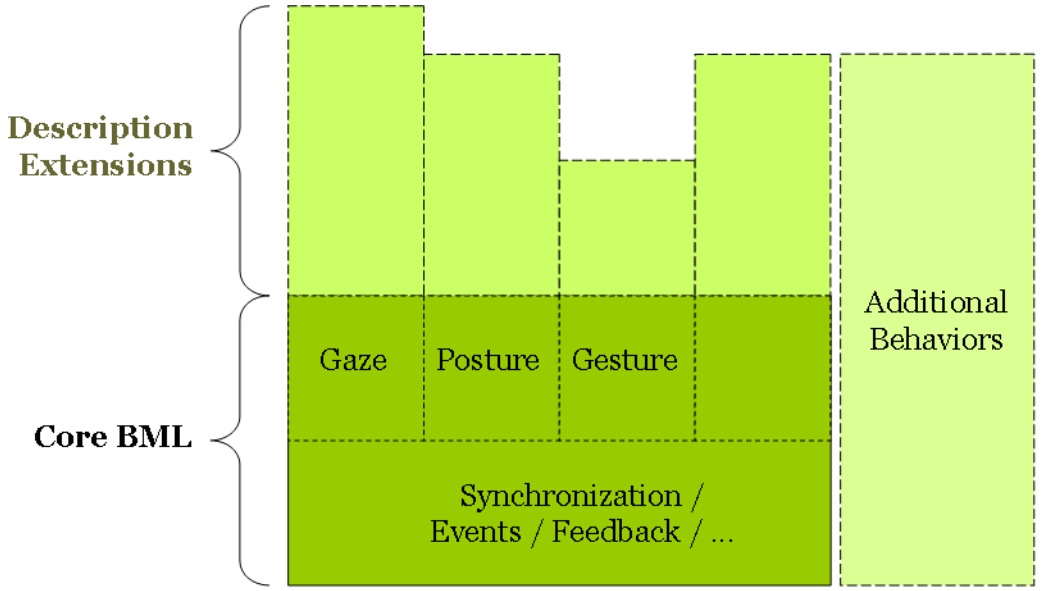


Figure 6.5: Extending BML. Description levels provide more detail on existing BML behaviors, additional behaviors are created as new XML elements (picture from `http://wiki.mindmakers.org/projects:bml:bml1.0-proposal`).

The type attribute of the description element should identify the type of content, indicating how it should be interpreted.

Description elements in BML can include existing representation languages such as SSML[7], MARY TTS[8], etcetera. or new languages can be created that make use of advanced realization capabilities. Each description element is a self-contained description of a behavior. It is also required that each description provides exactly the same synchronization points as its core BML description. It is however allowed to time the synchronization points in the description level at slightly different positions than those in the core BML. This can be used, for example, to provide synchronization at syllable level rather than at word level in a description extension of a speech behavior (see BML Example 6).

If a Realizer does not know 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.

[7] `http://www.w3.org/TR/speech-synthesis/`
[8] `http://mary.dfki.de/`

BML Example 5 Use an audio file to play back this speech behavior. If that is 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.

```
<speech id="s1">
   <text>This is a 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 a <emphasis>BML</emphasis> <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>
```

BML Example 6 In the Mary TTS description of speech behavior `s1`, the synchronization point `sync1` is defined to occur after the first syllable of "Hello". Since I cannot define sync points at syllable level in core BML, but the core BML does require the same sync points as the description, I define it to occur before "Hello" in the core description.

```
<speech id="s1">
  <text><sync id="sync1"/>Hello world!</text>
  <description priority="1" type="maryallophones">
    <maryxml xmlns="http://mary.dfki.de/2002/MaryXML">
    <p><s><phrase>
    <t accent="H*" g2p_method="lexicon" ph="h @ - ’ l @U" pos="UH"> Hello
    <syllable ph="h @">
      <ph p="h"/><ph p="@"/>
    </syllable>
    <mark name="sync1"/>
    <syllable accent="H*" ph="l @U" stress="1">
      <ph p="l"/><ph p="@U"/>
    </syllable>
    </t>
    <t accent="H*" g2p_method="lexicon" ph="’ w r= l d" pos="NN"> world
    <syllable accent="H*" ph="w r= l d" stress="1">
      <ph p="w"/><ph p="r="/><ph p="l"/><ph p="d"/>
    </syllable>
    </t>
    <t pos=".">!</t>
    <boundary breakindex="5" tone="L-L%"/>
    </phrase></s></p>
    </maryxml>
  </description>
</speech>
```

6.3 Recommendations

BML is a language that is still in development. Currently some of the behavior form specifications are unfinished (e.g. those of face) and the form description abstractions and parameters are not consistent over different behaviors. Behaviors can be synchronized using seven sync points of gesture, but the semantic meaning of these sync points is not specified clearly for other behaviors.

More importantly, several issues with behavior persistence and mutually exclusive behaviors are unresolved. Here I propose some solutions for these issues.

6.3.1 Persistent Behavior

The BML specification of a behavior does not need to include a specification of its end time. For most behaviors the BML Realizer automatically finds an appropriate end time. For some behaviors (currently `posture` and certain forms of `locomotion`) it makes sense to keep them running persistently if no end time is specified. This is used to makes a virtual human stay in a prescribed pose or keep him walking with a certain velocity. I recommend that gaze should be defined as persistent too, so that it can be used to keep the gaze on a target.

6.3.2 Behavior Persistence and the BML Block End

The notion of a BML block being finished is needed when sending its end feedback and when appending other BML blocks to it. If a BML block contains persistent behaviors, it is never finished. This means that once a BML block containing persistent behavior is sent to a Realizer, the append scheduling attribute is no longer meaningful. I recommend the inclusion of the BML^{T}/SmartBody append-after scheduling attribute (See Section 6.6) to the core BML standard to circumvent this issue.

Alternatively[9], behavior persistence could be replaced by behavior state and state transition descriptions. For example, rather than having a persistent posture behavior that retains the posture, a posture behavior could define the posture *change* and model the desired final posture state. This introduces state management as a responsibility of the Realizer. States could include the posture, the walking velocity, the gaze target, and so on. The execution of the behavior itself would then entail the state change (e.g. the change of gaze target, the change of walking velocity, the posture change); its end indicates that the state change was completed.

6.3.3 Mutually Exclusive Behaviors

For some behaviors it is impossible to have more instances of the same behavior active at the same time (for example: one cannot gaze at two objects at the same time, speak two sentences at the same time or stand in two poses at the same time). I define these behaviors as *mutually exclusive* behaviors.

[9] As suggested to me by Stefan Kopp.

For mutually exclusive behaviors that also allow behavior persistence, it makes sense to allow the latest behavior to overwrite previous behavior. This allows the virtual human to, for example, temporarily divert gaze to a new target and then return gaze to an earlier specified target. Such behavior suspension is currently proposed only for the `posture` behavior in core BML. I propose to extend this to the `gaze` behavior and to specify the exact overwriting rule as follows: the running behavior in a set of mutually exclusive behaviors is the behavior with the latest start time (earlier than the current time). If the same latest start time is shared by multiple behaviors, the active behavior is that with the earliest end time. If two or more mutually exclusive behaviors share the latest start and the earliest end time, all but one of them are dropped and the SAIBA Behavior Planner is informed of the dropped behaviors. Figure 6.6 illustrates this mutually exclusive behavior handling algorithm.

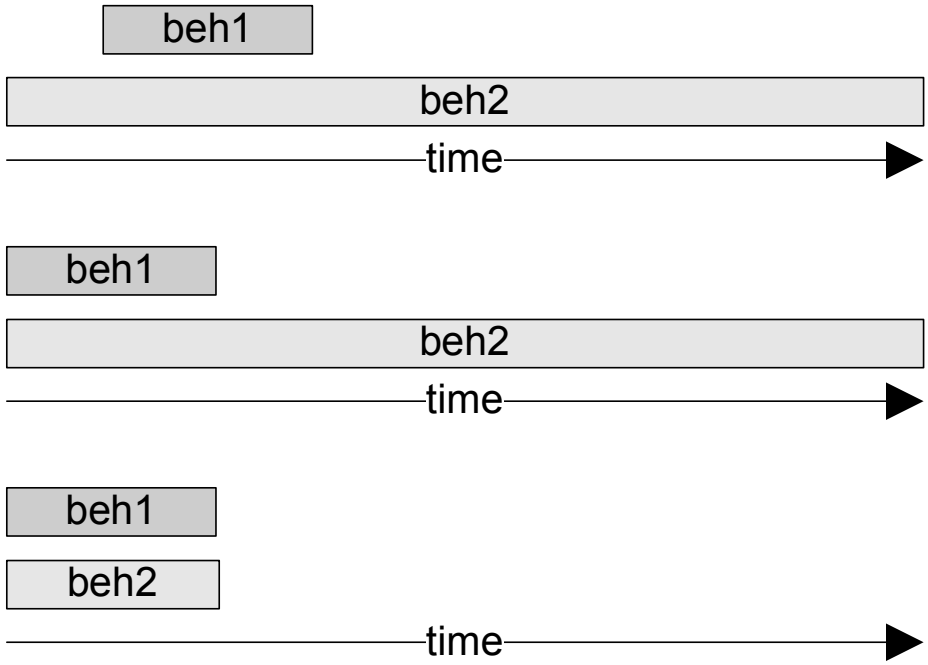


Figure 6.6: In all figures, behaviors `beh1` and `beh2` are two mutually exclusive behaviors. Top: `beh1` and `beh2` overlap during some interval of time. Since `beh1` has the latest start time, it will be selected to play during the overlapping time interval. Middle: `beh1` and `beh2` share the same start time. Since `beh1` has the earliest end time, it will be selected during the time interval on which the behaviors overlap. Bottom: `beh1` and `beh2` share the same start and end time, one of them is dropped and the SAIBA Behavior Planner is informed of this.

6.4 Continuous Interaction

The basic requirements for a specification language for multimodal behavior of a virtual human are clear. One should be able to specify short monologues of behavior

using different modalities (speech, face expression, gestures) and to specify the relative synchronization constraints between single behavior elements. Furthermore, one should be able to add additional behaviors, either to be performed immediately, in parallel with the earlier specified behavior, or to be appended after the other behavior is finished. BML satisfies such requirements.

In order to achieve the continuous interaction described in Chapter 5, there are four additional core requirements to such a markup language. It should allow the synchronization of ongoing behavior to predicted interlocutor events, flexible interruptibility, shape modification of ongoing behavior, and high responsiveness. In this section I will go into more detail on these four requirements.

Virtual humans should be able to synchronize their behavior with the behavior of an interlocutor or allow it to be aligned with predicted interlocutor events (such as the end of their speaking turns). This requires the specification of behavior alignment to predicted interlocutor events.[10] The time predictions of these events are updated incrementally, typically providing more detailed prediction accuracy at a later time. The multimodal behavior plan should be updated automatically, reflecting such an updated prediction.

Virtual humans should be capable of dealing with interruptions. Utterances may need to be (gracefully) broken off halfway through the sentence because the user interrupts the virtual human. A gesture that was already initiated may have to be be abandoned (retracted) directly after the preparation or hold phase because it is suddenly no longer relevant (e.g., the object to be pointed at has disappeared, or the user is no longer looking at the virtual human). However, state-of-the-art virtual human systems do not yet allow one to specify exactly where to interrupt ongoing behavior. Furthermore, one also needs to specify alternative continuations for the interrupted behavior: the abandoned gesture needs to be retracted; the last word in the interrupted sentence may have to be pronounced in a 'fade-out' manner (e.g., with a falling intonation and softer voice); et cetera. In this section, I present a mechanism to specify graceful interruptions of ongoing behavior (speech, gestures) that allows one to specify exactly where the behavior needs to be interrupted, and, in addition, to specify how to alternatively and immediately continue after the interruption.

Virtual humans need mechanisms to slightly modify the shape of ongoing behavior (e.g., increase gesture amplitude or speech loudness). Such shape modifications typically occur in reaction to interlocutor events (for example, the interlocutor trying to take the speaking turn).

Virtual humans should exhibit quick and immediate responsiveness to the user. In human-human interaction, some responses are delivered with a short (500 milliseconds) to near-zero delay. The coordination between humans in synchrony and alignment behavior occurs on a similar timescale. For a virtual human this means that, given a certain action from the user, specification of new behavior and changes to the ongoing behavior must be done with virtually no delay at all (see also Chapter 5). Given the fact that behavior scheduling takes a non-negligible amount of

[10] See also Chapter 5.5, for a more detailed explanation of why this should be in the specification itself rather than be handled through the Behavior Planner.

time, it should be possible to specify potential responses by the virtual human ahead of time, to be activated with near-zero delay at the appropriate moment. This also argues for rapid handling of the requirements of shape modification and alignment to predicted interlocutor events, preferably in such a manner that time-consuming rescheduling is avoided.

This thesis contributes the BML extension BML Twente that augments BML to satisfy the four fundamental requirements for the specification of continuous interaction mentioned above.

6.5 Scenarios for Continuous Interaction

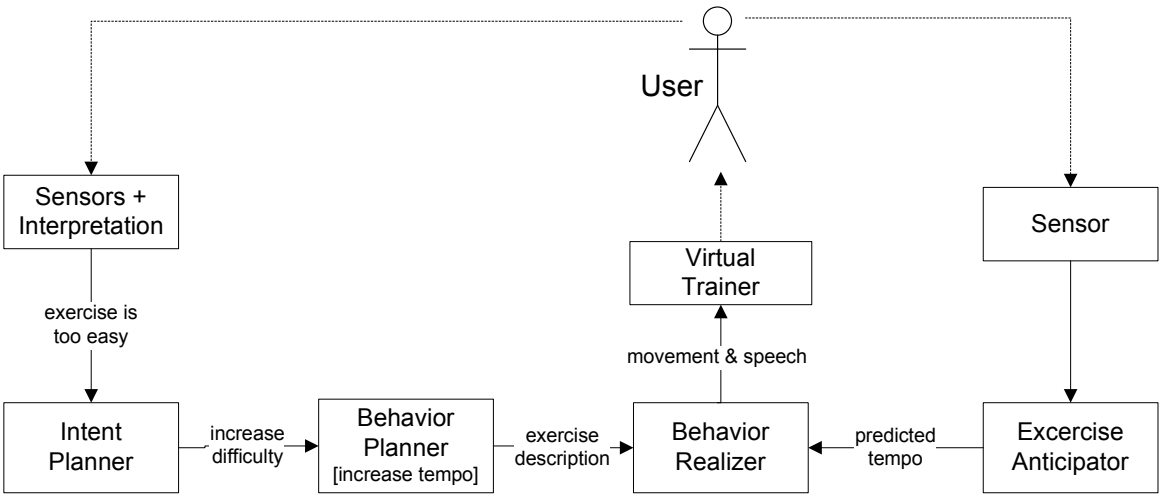
Continuous interaction is often observed in fluent turn-taking or when humans exercise the same behavior together. This section provides some typical scenarios of continuous interaction and demonstrates how they can be specified in BML^{T}.

6.5.1 Interpersonal Behavior Synchronization

A Virtual Trainer is exercising together with a user. The Virtual Trainer infers from sensor data that the exercise is too easy for the user. She decides to increase the difficulty of the exercise. One way of doing this is by increasing the exercise tempo. A subtle technique to make the user move faster is to move in the same tempo as the user but slightly ahead of him, so he constantly has the feeling of being 'too late' in his movements. The movements of the user are observed with a sensor. An Anticipator has been designed that can perceive the tempo a user is exercising in using this sensor, and from this information extrapolates future exercise time events (see also Chapter 10.3.6). By making use of the time predictions from this `exerciseAnticipator`, one can specify the trainer's movement to be slightly ahead of them.

BML Example 7 illustrates this virtual trainer scenario. Based on sensor data, the exercise Anticipator provides synchronization points for predicted events in the exercise (e.g. when will the next jumpstart or squatdown occur). This information can be used as synchronization constraints in a BML block. The BML block in BML Example 7 describes how synchronization points of a procedural exercise animation (`exercise1`) are synchronized to be slightly ahead (0.5 seconds) of the anticipated synchronization points in the exercise as executed by the user. `exercise1:squatdown` and `exercise1:jumpstart` refer to the squat down position and the start of the jump in the squat-jump exercise animation respectively. `exerciseAnticipator:squatdown` and `exerciseAnticipator:jumpstart` refer to the anticipated timing of squatdown and jumpstart as predicted by movement of the user. As the sync points used here are custom sync points (rather than the standard sync points listed in Figure 6.3) in both the animation and the Anticipator, the `synchronize` element is used to define a time constraint between them.

BML Example 7 Using an exercise Anticipator to synchronize the movement of the Virtual Trainer with that of a user.




```
<bmlt:procanimation id="exercise1" name="squat-jump"/>
<constraint id="c1">
   <synchronize ref="exercise1:squatdown">
    <sync ref="anticipators:exerciseAnticipator:squatdown-0.5"/>
   </synchronize>
   <synchronize ref="exercise1:jumpstart">
    <sync ref="anticipators:exerciseAnticipator:jumpstart-0.5"/>
   </synchronize>
   ...
</constraint>
```

6.5.2 Turn-Taking

The following examples show how conversational turn-taking strategies can be embedded in the SAIBA framework and how Anticipators and BML^T's interrupt and parametervaluechange behaviors allow their elegant specification.

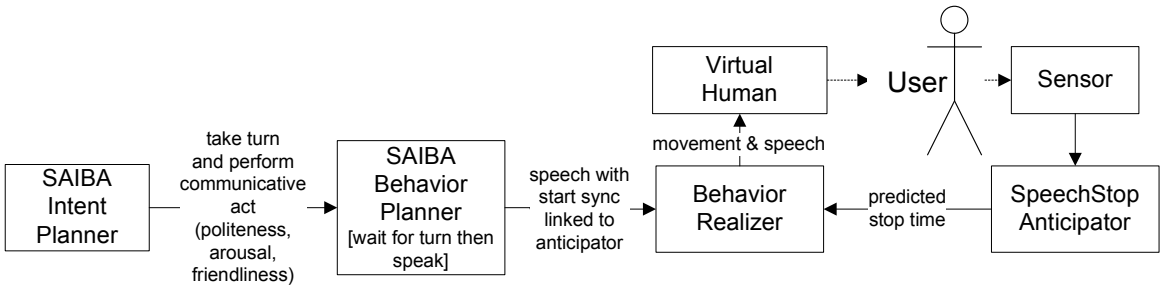
6.5.2.1 Taking the Turn

Humans can take the turn at different moments, for example, slightly before their interaction partner stops speaking, at exactly the moment their interaction partner stops speaking, or slightly after their interaction partner stops speaking. The turn-taking strategy used can modulate the impression of assertiveness, agreeableness, conversational skill, politeness, friendliness and arousal of the virtual human (see [179, 181], Chapter 5.1.3). I assume that one can design an Anticipator that can predict the end of speech of a user (see Chapter 12.2.2 for possible implementations), called the `speechStopAnticipator`.

In BML Example 8, the SAIBA Intent Planner decides to take the turn and perform a communicative act. The SAIBA Behavior Planner selects a turn taking strat-

egy, based on, for example, the politeness, arousal and friendliness of the virtual human. In the illustrated case, it waits for the user to stop speaking and starts speaking after a certain delay x (note that x could be negative to start speaking slightly before the user stops speaking). The SAIBA Behavior Planner therefore only specifies that the virtual human starts speaking after the user stops speaking, and the exact and precisely timed execution of this behavior is handled by the Behavior Realizer, using the `speechStopAnticipator`.

BML Example 8 Taking the turn.




```
<bml id="bml1">
  <speech id="speech1" start="anticipators:speechStopAnticipator:stop+x">
    <text>Bla bla</text>
  </speech>
</bml>
```

Turn-taking using Anticipator timing offers several improvements over having a the SAIBA Behavior Planner send the BML block that takes the turn to the Realizer at the moment the turn should be taken: 1) if the BML block is sent 'early enough', turn-taking using an Anticipator avoids or at least decreases scheduling delays and allows the block to be executed exactly at its desired time, 2) as long as `speech1` is not started, its desired starting time can be changed, providing behavior execution that can make use of incrementally improving turn start timing predictions, 3) by providing alignment to Anticipators as a general Realizer mechanism, the design of SAIBA Behavior Planners is simplified.[11]

6.5.2.2 Keeping the Turn

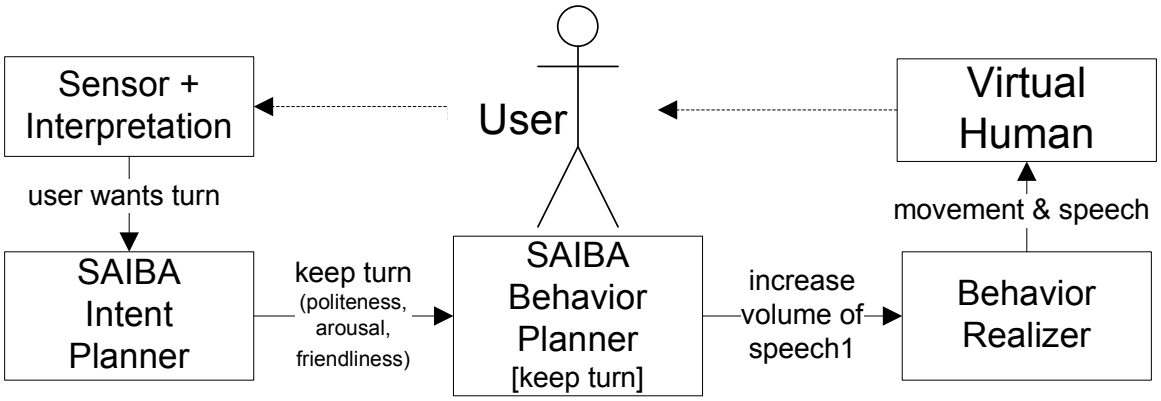
In BML Example 9 the SAIBA Intent Planner is informed by an interpretation of sensor values that the user would like to get the turn. The SAIBA Intent Planner decides that the virtual human would like to keep the turn. Based on the provided politeness, arousal and friendliness values, the SAIBA Behavior Planner decides to realize this in intent by increasing the volume of behavior `speech1` (from its starting

[11]This of course means that the Realizer becomes more complex. However, one typically builds only one Realizer, but several application/experiment specific SAIBA Behavior Planners that require such functionality (see also Chapter 8.1 and Chapter 10).

value to target value 90 along a linear trajectory, the target value is to be reached 1 second after the custom `sync` synchronization point in `speech1`).

This is achieved by sending a new BML block to the BML Realizer that contains a `parametervaluechange` behavior targeting the volume of `speech1`. The parametervaluechange is synchronized with the `speech1` behavior of BML block `bml1`. Normally BML does not allow such synchronization with behaviors in an external BML block. BML^{T} provides the `allowexternalrefs` attribute to indicate that, within a BML block, external time references are allowed.

BML Example 9 Keeping the turn.




```
<bml id="speechchange" bmlt:allowexternalrefs="true">
  <bmlt:parametervaluechange target="bml1:speech1" paramId="volume"
    start="bml1:speech1:sync1" stroke="bml1:speech1:sync1+1">
    <bmlt:trajectory type="linear" targetValue="90"/>
  </bmlt:parametervaluechange>
</bml>
```

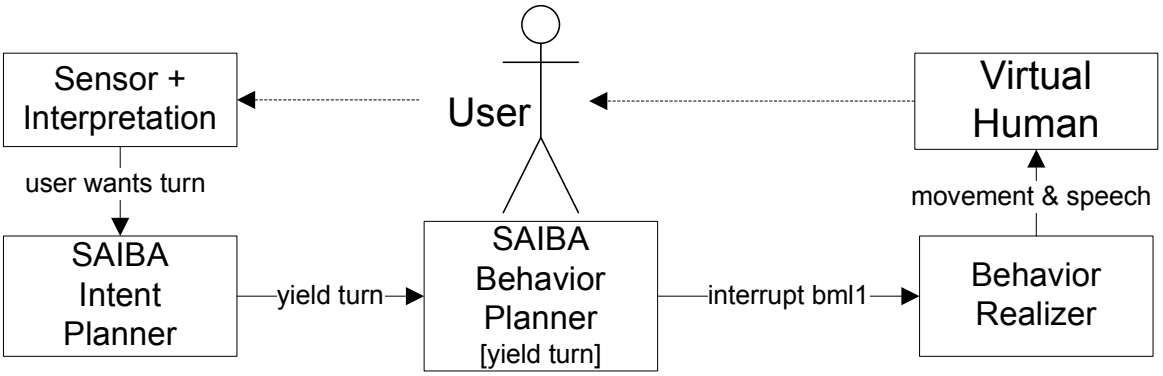
6.5.2.3 Turn Yielding

In BML Example 10 the SAIBA Intent Planner is informed by an interpretation of sensor values that the user would like to get the turn. The SAIBA Intent Planner decides that the virtual human should yield its turn. The current utterance is specified in `bml1` and contains behaviors `speech1` and `gesture1`. Depending on where the Realizer is in executing `bml1`, the SAIBA Behavior Planner employs different interruption strategies:

1. If `gesture1` has not been started, all behavior in `bml1` is interrupted using the BML block in BML Example 10a.

2. If `gesture1` is already in its retraction phase, or has already finished, there is no need to interrupt it, and, using the BML block in BML Example 10b, only `speech1` is interrupted.

BML Example 10 Yielding the turn.



(a) Yield the turn by interrupting all behavior in `bml1`.

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

(b) Yield the turn by interrupting all behavior in `bml1` excluding `gesture1`.

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

(c) Yield the turn by interrupting all behavior in `bml1`. Insert a behavior (`relaxArm`) that gracefully moves the gesturing arm back to its rest position.

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

3. If `gesture1` has started, but is not yet in its retraction phase, interrupt all behavior in `bml1` and insert a behavior (`relaxArm`) that gracefully moves the gesturing arm back to its rest position. This is achieved using the BML block in BML Example 10c.

Note that this example requires that the SAIBA Behavior Planner keep track of the behavior sent to the Realizer and monitors its execution progress. The latter is achieved using the feedback messages that are sent to the SAIBA Behavior Planner by the Realizer (see also Section 6.2.4). A generic module within the SAIBA Behavior Planner could be designed this often occurring scenario of gracefully interrupting a piece of speech with an aligned gesture.

6.5.3 Incremental Interpretation and Generation

Skantze and Hjalmarsson [271] designed Jindigo, an incremental speech recognition and generation architecture that facilitates opportunistic speech planning (see also Chapter 5.1.3.2), using incremental speech recognition and generation.

Their speech generation is specified using SpeechPlans, which is a directed graph of SpeechUnits. The SpeechPlan is generated on the fly. It can capture multiple different realization options. A path on the graph is selected at run time by an ActionManager. This ActionManager could be seen as a SAIBA Behavior Planner, and the SpeechPlan could be expressed in BML^T. This allows such an incremental generation paradigm to be used within the SAIBA framework. The scenario of BML Example 11, taken from [271] illustrates this.

The BML^T pre-planning and activation capabilities allow the construction of a directed graph of SpeechUnits in the multimodal plan of the BML Realizer. The SpeechPlan is built incrementally in BML. It is extended as more input from an automatic speech recognizer becomes available. First, two alternative apositional turn beginnings are created using `bml1` and `bml2`. These apositional turn beginnings can be executed as soon as the virtual human would like to take the turn, even without having a plan for the rest of its sentence at hand. As soon as the partial sentence "how much" is recognized, the Action Manager adds some alternatives describing cost to the plan using `bml3`, `bml4` and `bml5`. When the sentence "how much is the doll" is fully parsed, the ActionPlanner can add a sentence describing the cost, using `bml6`. Figure 11c and 11d show how the paths "eh, it costs 40 crowns" and "let's say 40 crowns" are selected respectively, using BML^T. The selection of a node on the graph involves sending a BML block that activates its pre-planned BML block and interrupts (removes) all alternative nodes.

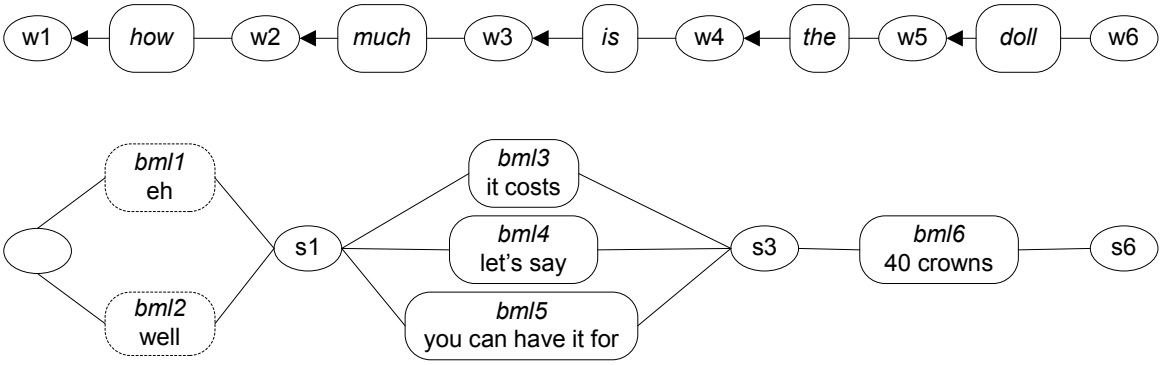
Feedback on the progress of the BML *scheduling* process is essential in such incremental generation. Elckerlyc provides feedback indicating when a BML block is scheduled (see Section 6.7.1). This feedback may be used by the Action Manager to select a path on the graph. For example, if Elckerlyc has not yet finished scheduling `bml3`, `bml4` and `bml5`, the Action Manager can select an apositional beginning (e.g. `bml1` or `bml2`) to take the turn while not having a SpeechPlan at hand right away. Alternatively, if `bml3`, `bml4` and `bml5` are scheduled, the apositional beginning of the SpeechPlan (containing either `bml1` or `bml2`) may be omitted. The Action Manager might be forced to select a node when not all alternatives have been scheduled. It can then simply select one that is scheduled, removing all scheduled alternatives and canceling scheduling of the unscheduled ones.

In a dialog system that uses incremental processing, input hypotheses might be revised, which could lead to revisions in the ongoing SpeechPlan. To allow this, the Realizer must support self-repairs (see Figure 6.7). These repairs may be covert (they are achieved by changing planned behavior without the interlocutor noticing the plan change) or overt (involving an explicit correction). Overt revisions may include an apositional beginning (e.g. sorry, that's wrong). To decide whether to use an overt or covert correction and exactly which one, the ActionManager needs to know which SpeechUnits have been executed already. As in the scenario of Section 6.5.2.3 this requires that the ActionManager keeps track of the behavior sent to the Realizer and monitors its execution progress. The latter is achieved using the feedback messages that are sent to the dialog manager by the Realizer (see also Section 6.2.4). Thus, these self-repair mechanisms can be specified using BML^T.

Self-repair requires the interruption of ongoing speech, using a similar BML^T

BML Example 11 Setting up and executing an incremental SpeechPlan.

(a) Top: recognition from an automatic speech recognizer, bottom: the produced SpeechPlan. Vertex s1 is associated with w1, s3 with w3, and so on. (Figure based upon Figure 4 in [271], bml ids are mine).



(b) Incrementally setting up the SpeechPlan, using BML^{T}

After w1:

```
<bml id="bml1" bmlt:preplan="true">
  <speech id="s1"><text>eh</text></speech>
</bml>
<bml id="bml2" bmlt:preplan="true">
  <speech id="s1"><text>well</text></speech>
</bml>
```

After w3:

```
<bml id="bml3" scheduling="append-after(bml1,bml2)" bmlt:preplan="true">
  <speech id="s1"><text>it costs</text></speech>
</bml>
<bml id="bml4" scheduling="append-after(bml1,bml2)" bmlt:preplan="true">
  <speech id="s1"><text>let's say</text></speech>
</bml>
<bml id="bml5" scheduling="append-after(bml1,bml2)" bmlt:preplan="true">
  <speech id="s1"><text>you can have it for</text></speech>
</bml>
```

After w6:

```
<bml id="bml6" scheduling="append-after(bml3,bml4,bml5)" bmlt:preplan="true">
  <speech id="s1"><text>40 crowns</text></speech>
</bml>
```

(c) Realizing the path "eh, it costs 40 crowns"

```
<bml id="bml7" bmlt:interrupt="bml2,bml4,bml5" bmlt:onStart="bml1,bml3,bml6"/>
```

Or, incrementally:

```
<bml id="bml7" bmlt:interrupt="bml2" bmlt:onStart="bml1"/>
<bml id="bml8" bmlt:interrupt="bml4,bml5" bmlt:onStart="bml3"/>
<bml id="bml9" bmlt:onStart="bml6"/>
```

(d) Realizing the path "let's say 40 crowns"

```
<bml id="bml7" bmlt:interrupt="bml1,bml2,bml3,bml5" bmlt:onStart="bml4"/>
<bml id="bml8" bmlt:onStart="bml6"/>
```

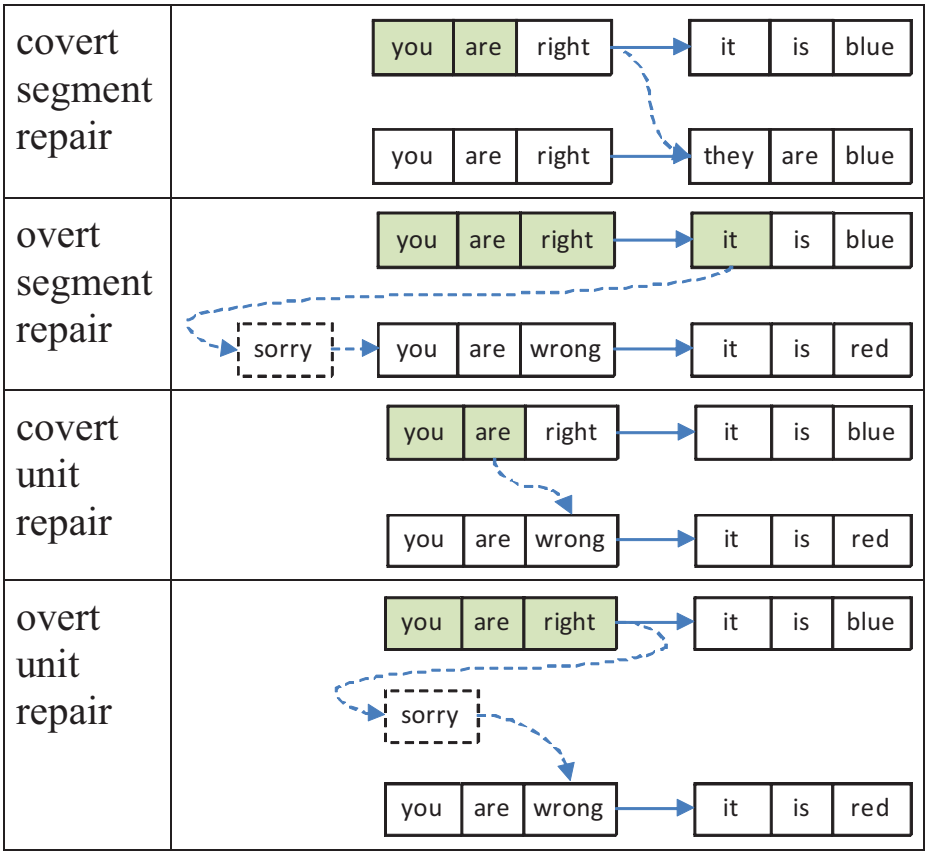


Figure 6.7: Different types of self-repairs. The shaded boxes show which SpeechUnits have been realized, or are about to be realized, at the point of revision. Figure from [271].

specification as in the scenario in Section 6.5.2.3. The construction of new paths on SpeechPlan, including optional apositional beginnings is illustrated in the BML Example 11.

6.6 BML^{T}

I developed the BML extension BML^{T} (BML Twente) specifically to accommodate the specification requirements of continuous interaction. In addition to that, it provides several behaviors and description extensions that are specific to Elckerlyc. In this thesis, the prefix `bmlt` is used for all BML^{T} specific elements.[12]

[12]For clarity purposes I omit the BML^{T} namespace declaration in all examples in this chapter.

6.6.1 Pre-planning and Activation

Scheduling a BML block typically takes a non-negligible amount of time, especially if the timing of speech is to be obtained through speech synthesis software.[13] This is problematic for developing highly responsive virtual humans. BML^T provides *pre-planning* as a mechanism to construct a behavior plan that can be activated later on. In a typical usage scenario of pre-planning, the SAIBA Behavior Planner already knows what behavior to execute, and wants to have it ready for (near) instant execution, for example in reaction to some event such as an incoming response from the user. Pre-planning is set up for a BML block, using the BML^T pre-plan attribute in that block. Pre-planned BML blocks can be activated using a BML^T activate behavior. The pre-planned BML block is activated as soon the activate behavior starts. BML Example 12 illustrates the BML used for pre-planning.

BML Example 12 Several BML blocks illustrating the pre-planning and activation of pre-planned behavior.

(a) Pre-plan bml1.

```
<bml id="bml1" bmlt:preplan="true">
  ...
</bml>
```

(b) Activate pre-planned behavior bml1.

```
<bml id="bmlX">
  <bmlt:activate id="a1" target="bml1"/>
</bml>
```

(c) Activate pre-planned behavior `bml2` after `nod1` is finished.

```
<bml id="bml1">
  <head id="nod1" action="ROTATION" rotation="SHAKE"/>
  <bmlt:activate start="nod1:end" target="bml2"/>
  ...
</bml>
```

Pre-planning on its own is not new. Kopp and Wachsmuth [155] make use of incremental (pre)planning mechanisms for the purpose of late planning of transitions between segments of gesture and speech, which are highly context dependent (depending on current gesture and the next gesture), but for which some parts can be pre-constructed (e.g. the speech synthesis). The Sensitive Artificial Listener (SAL) system [258] can pre-plan behavior and activate pre-planned behavior at will, like Elckerlyc's Scheduler. The SAL system does this by providing a separate pre-planning scheduler and a preplan activation trigger which are both outside the normal BML stream. Unlike the BML based pre-planning mechanisms used within

[13] Providing timing information on a short (11 word) sentence takes 110-220ms on the TTS systems used in Elckerlyc; a within one video frame scheduling delay will not be achieved for the next 8 years with current TTS systems (given Moore's law).

Elckerlyc's Scheduler, their setup does not allow a direct specification of the (timing) relations between the normally planned and pre-planned behavior. Example 12c illustrates one such relation that BML^{T} allows: the pre-planned BML blocks bml1 is started after `nod1` is finished. In Section 6.6.4, I show how a specification of timing relations between normally planned and pre-planned blocks can be used to achieve a gracious interruption with an immediate alternative continuation.

6.6.2 BML Procedures

Several applications, including routinized speech and apositional turn beginnings (see Chapter 5.1.3), can profit from the instant use of already scheduled behaviors. Pre-planning allows one to use an already scheduled BML block only once, after it is activated the block cannot be reused. It would be beneficial to allow the reuse (that is: use it more than once) of, for example, an already scheduled apositional beginning in a dialogue.

Such reuse could be facilitated by BML procedures. Here I illustrate the concept[14] of BML procedures with a possible specification for them. A `defineproc` XML block defines a BML procedure: a set of behaviors and their constraints. This procedure can be called using `callproc`. Conceptually, `callproc` constructs a new BML block on the basis of a proc id, bml id and scheduling attribute and sends this to the Realizer (See BML Example 13). Because the behaviors in the BML block were already fully scheduled with their procedure definition, their execution is started instantly.

BML Example 13 Defining and using BML procedures.

(a) Defining proc1.

```
<bmlt:defineproc id="proc1">
  <gaze type="AT" id="gaze1" target="AUDIENCE"/>
  <speech start="gaze1:ready" id="speech1">
    <text>Welcome ladies and gentlemen!</text>
  </speech>
</bmlt:defineproc>
```

(b) Using proc1 as bml1.

```
<bmlt:useproc procid="proc1" bmlid="bml1" scheduling="append"/>
```

(c) The resulting BML block.

```
<bml id="bml1" scheduling="append">
  <gaze type="AT" id="gaze1" target="AUDIENCE"/>
  <speech start="gaze1:ready" id="speech1">
    <text>Welcome ladies and gentlemen!</text>
  </speech>
</bml>
```

[14]The functionality discussed here is not implemented in Elckerlyc yet.

Note that in this specification, the procedure definition and calling is not a part of BML itself. Rather, it forms a convenient shorthand for the SAIBA Behavior Planner to ensure that a BML block is executed without a scheduling delay. This shorthand requires an additional communication channel between the Realizer and the SAIBA Behavior Planner to define and call procedures. Alternatively, the procedure calling could be integrated into BML, thus allowing a BML block to 'call a procedure'. Such procedure calling could then potentially replace BML^T's pre-planning and activation functionality. However, pre-planning and procedure definition / calling offer different performance trade-offs. Procedures avoid the additional scheduling time of scheduling a behavior more than once. Pre-planning however avoids the memory footprint of procedures that are used only once. It might thus be worth the (Realizer) implementation effort to allow both.

6.6.3 The Interrupt Behavior

The BML^T interrupt behavior provides the capability of specifying precisely when specific running or scheduled behaviors should be interrupted. A simple example would be to start a "look-at" behavior by the virtual human, while it is speaking (in the example through BML block `bml1`), and to interrupt the speech behavior as soon as the "look-at" behavior has finished (see BML Example 14).

BML Example 14 Interrupt `bml1` as soon as `gaze1:ready` is reached

```
<bmlt:interrupt id="i1" start="gaze1:ready" target="bml1"/>
```

As soon as the BML^T interrupt behavior executes it interrupts a complete BML block, referred to as the 'target'. Interrupts are normal BML behaviors, so they have standard BML attributes such as an `id` or `start` sync point, and can be synchronized with other behaviors as usual.

It is often desirable to interrupt only selected behaviors in a BML block. For example, typically it is undesirable to interrupt gestures that are already in their retraction phase. The `include` and `exclude` attributes of the `interrupt` behavior allow this (see BML Example 15).

BML Example 15 Interrupt all behavior in `bml1`, with the exception of `gesture1`. `speech1` is interrupted at the time of its synchronization point `sync1`. All other behavior in `bml1` is interrupted at `shake1:stroke`.

```
<bmlt:interrupt id="i1" target="bml1" start="shake1:stroke"
  exclude="speech1,gesture1"/>
<bmlt:interrupt id="i2" target="bml1" include="speech1"
  start="bml1:speech1:sync1"/>
```

The SmartBody Realizer [280] provides an interrupt behavior that has similar functionality to BML^T's interrupt behavior. It does however not allow the selective

interruption of behaviors in a BML block. I have based the syntax of the BML^T interrupt behavior on the specification of the interrupt behavior used in the SmartBody BML Realizer.

6.6.4 Instantaneous Gracious Interruption: Combining Interruption and Pre-planning

Interrupt behaviors can be combined with activate behaviors to provide instantaneous, but gracious interruption (see BML Example 16).

BML Example 16 The Realizer interrupts all behaviors in `bml1`. `speech1` is interrupted at `sync1` and gracefully ended with some trailing speech using `bml3`, `gesture1` is interrupted at its `stroke_start`, and followed by the content of `bml4`. All other behaviors in `bml1` are interrupted at the start of `i1` (that is, at `shake1:stroke`).

```
<bmlt:interrupt id="i1" target="bml1" start="shake1:stroke"
  exclude="speech1,gesture1"/>
<bmlt:interrupt id="i2" target="bml1"include="speech1" start="bml1:speech1:sync1"/>
<bmlt:interrupt id="i3" target="bml1" include="gesture1"
  start="bml1:gesture1:stroke_end"/>
<bmlt:activate start="bml1:speech1:sync1" id="a1" target="bml3"/>
<bmlt:activate start="bml1:gesture1:stroke_start" id="a2" target="bml4"/>
```

6.6.5 Anticipators

The Anticipator allows behavior to be synchronized to predicted events of an interlocutor or other outside world events. For example, an Anticipator could be designed to predict the end of a turn of the interlocutor, or to predict the tempo of a user that does an exercise with a virtual trainer.

An Anticipator instantiates synchronization points that can be used in the BML stream to constrain the timing of behaviors. It uses perceptions of events in the real world to continually update the timing of these constraints, by extrapolating the perceptions into *predictions* of the timing of future events. Such an update typically concerns a micro-adjustment in the timing of some behavior in the behavior plan. The Anticipator allows such changes to occur without time consuming rescheduling of the behavior plan.

BML Example 17 illustrates the use of an Anticipator that predicts when the interlocutor will stop speaking. Behavior `speech1` is started 1 second after that event. The timing of `speechStopAnticipator:stop` can be updated continually with predictions of increasing precision of the interlocutors turn stop time (provided that `speech1` has not started yet). Such updates automatically adjust the start time of `speech1`.

BML Example 17 Taking the turn.

```
<bml id="bml1">
  <speech id="speech1" start="anticipators:speechStopAnticipator:stop+1">
    <text>Bla bla</text>
  </speech>
</bml>
```

6.6.6 Specifying Parameter Value Changes

The `parametervaluechange` behavior allows the modification of certain parameter values of ongoing behavior without requiring a complete rebuild of the behavior plan. For example, a `parametervaluechange` can be used to make a virtual human speak louder in order to keep the turn if her interaction partner tries to interrupt her (see also the scenario in Section 6.5.2.2).

The `parametervaluechange` behavior specifies how a certain parameter value within one behavior is to be set to a new target value. The `trajectory` element allows a precise specification of the trajectory the parameter value follows to achieve its target value (e.g. linear, ease-in-ease-out, or instantly). By using a BML behavior to change parameter values, we have access to the BML synchronization mechanisms. That is: the timing of the parameter change can be specified precisely in relation to the timing of targeted behaviors or to other behaviors. An initial value for the parameter can be specified in the trajectory, using the `initialValue` attribute. If the initial value is not explicitly defined, the `parametervaluechange` behavior constructs a trajectory starting at the parameter value obtained from the targeted behavior, at the start time of the `parametervaluechange` behavior. The `parametervaluechange` behavior sets the parametervalue on the target behavior according to the trajectory, in such a way that the `targetValue` is reached at the stroke of the `parametervaluechange` (see also Figure 6.8). Constraining the end sync of the `parametervaluechange` behavior is not allowed: it ends automatically with its target behavior. In its phase between stroke and end, the `parametervaluechange` retains the `targetValue`.

The Multimodal Presentation Markup Language[42] provides a setStateParameter action which is similar to the `parametervaluechange` behavior. Such a setStateParameter action defines the new value of a parameter, and when it is to be achieved. The setParameterValue action addresses changes in global parameters values such as the arousal of the virtual human. These parameter values are automatically bound to local parameter values on *all* ongoing and scheduled behaviors (e.g. arousal could change both the amplitude of ongoing/scheduled beat gestures and the volume of ongoing/scheduled speech). In the SAIBA framework, the mapping of such global parameter changes to local parameter changes would be handled by the Intent and/or the SAIBA Behavior Planner rather than the Realizer. BML^T provides the SAIBA Behavior Planner with specification mechanisms to target the local parameter value changes within *one* ongoing or scheduled behavior in a very detailed manner.

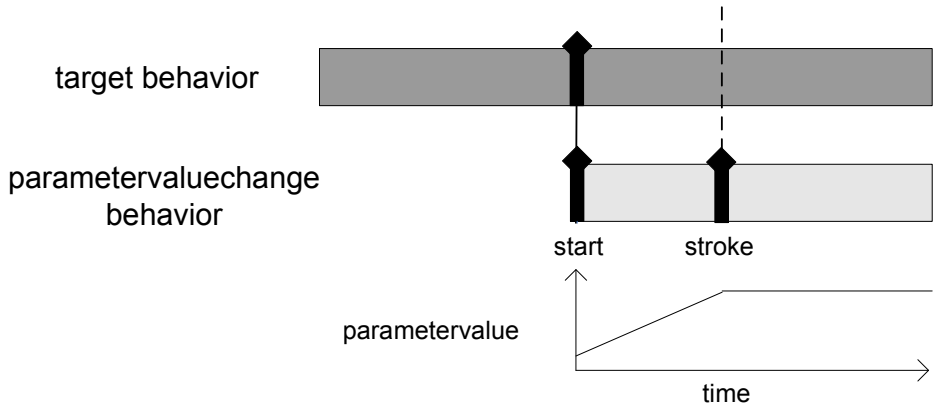


Figure 6.8: The start of a `parametervaluechange` behavior is linked to a sync in a target behavior, its stroke is linked to some other sync point (not shown in figure). The parameter value is changed along a linear trajectory from start to stroke. The target value is retained after the stroke.

6.6.7 Other BML^T Behaviors and Descriptions

Besides the behaviors facilitating continuous interaction, BML^T offers several other behaviors:

- `audiofile`: plays an audiofile, specified by a filename.
- `controller`: a physical controller, specified by its Java class name and several controller specific parameter values. A controller behavior always describes a desired state that the virtual human is supposed to retain rather than set of motion phases it has to move through. It is therefore implemented as a persistent behavior.
- `keyframe`: Describes a keyframe (or mocap) animation. The 'joints' parameter can be used to select the joints the animation should act upon. This is useful to select motion on a subsection of the body from full body mocap. The 'mirror' parameter can be used to mirror the animation in the mid-sagittal plane, allowing, for example, motions with the right arm to be reused on the left arm.
- `procanimation`: a procedural animation, specified by its filename and several custom parameter values, including joint selection and mirroring as described above.
- `transition`: a transition animation. The start and end pose of the transition are determined automatically from its surrounding motions, the transition is specified only by the joints it is to act upon and the transition type (e.g. SlerpTransition, HermiteSplineTransition). Typically transition behaviors are used to connect procedural animations or to form the preparation phase of a gesture that does not contain one.
- `noise`: generates a noise (typically Perlin noise) animation, specified by the type of noise and custom parameters.

- `facemorph`: a morph based face animation, specified by the morph target and its intensity.

Note that several BML^T behaviors have parameters that vary with their specific procedural animation, controller, and so on. Such custom parameters are configured as name, value pairs using the BML^T `parameter` tag (see also BML Example 18).

BML Example 18 A balance controller with pelvis height at 0.9 times leg length (so slightly bent knees) and slightly increased stiffness, used as an independent BML^T behavior (top) or as a description extension of a posture behavior (bottom).

(a) The BML^T controller behavior.

```
<bmlt:controller id="balance1" class="BalanceController">
  <bmlt:parameter name="pelvisheight" value="0.9"/>
  <bmlt:parameter name="stiffnessmultiplier" value="1.1"/>
</bmlt:controller>
```

(b) A BML^T controller used as description extension in a BML posture behavior.

```
<posture id="balance1" part="lower" stance="standing" shape="open">
  <description priority="1" type="controller">
    <bmlt:controller class="BalanceController">
      <bmlt:parameter name="pelvisheight" value="0.9"/>
      <bmlt:parameter name="stiffnessmultiplier" value="1.1"/>
    </bmlt:controller>
  </description>
</posture>
```

Most BML^T behaviors may also be used as a description extension for a core BML behavior. For example, `procanimation` can be used as description extension for `gesture` and `controller` can be used as description extension for `posture` (see also BML Example 18). BML^T also supports several speech description extensions, including SSML, Microsoft SAPI and various MaryTTS specifications.

BML Example 19 The persistent controller that lets the left arm hang down loosely (`larm`) is temporarily overwritten by an NoController (`larm1`).

```
<bmlt:controller id="larm" class="CompoundController" name="leftarmhang">
  <bmlt:parameter name="replacementgroup" value="leftarm"/>
  <bmlt:parameter name="shoulder:anglez" value="0.3"/>
  <bmlt:parameter name="shoulder:ksz" value="40"/>
</bmlt:controller>
<bmlt:controller id="larm1" class="NoController" start="g1:start" end="g1:end">
  <bmlt:parameter name="replacementgroup" value="leftarm"/>
</bmlt:controller>
```

Each BML^T behavior can be set up as a mutually exclusive behavior by using its `replacementgroup` parameter. If two BML^T behaviors have the same value for their `replacementgroup` parameter, they are mutually exclusive. If a BML^T behavior

is used as a description extension of a mutually exclusive core BML behavior, it typically takes the `replacementgroup` of that behavior. BML Example 19 shows a typical mutual exclusion scenario: the persistent controller that lets the left hang down loosely is temporarily overwritten by a 'NoController' that releases the control of the left arm. This allows the left arm to be used for a gesture.

6.6.8 BML^T BML Attributes

In addition to the `preplan` attribute, BML^T provides several other attributes that are applied in the `bml` element itself.

6.6.8.1 The Append-after Scheduling Attribute

BML^T adds the append-after(X) scheduling attribute in addition to the mandatory BML scheduling attributes. X is a comma separated list of prior BML block ids. The append-after(X) scheduling attribute instructs the Realizer to execute the new BML block immediately after all behaviors from the prior BML blocks on list X have finished (see BML Example 20).

BML Example 20 BML block `bml4` is appended after `bml2` and `bml3`.

```
<bml id="bml4" scheduling="append-after(bml2,bml3)">
  ...
</bml>
```

6.6.8.2 The allowexternalrefs Attribute

BML^T's `allowexternalrefs` attribute is used to indicate that a BML block may contain time constraints that refer to behaviors in other (external) BML blocks. Such references are of the form `bmlid:behaviorid:syncid`. BML Example 21 illustrates this.

BML Example 21 BML block `bml2` contains a `parametervaluechange` behavior that modifies behavior `speech1` in `bml1`. Two time constraints link the timing of `pvc1:start` en `pvc1:stroke` sync points to the external sync points `bml1:speech1:sync1` and `bml1:speech1:end` respectively.

```
<bml id="bml2" bmlt:allowexternalrefs="true">
  <bmlt:parametervaluechange id="pvc1" target="bml1:speech1"
   paramId="volume"  start="bml1:speech1:sync1" stroke="bml1:speech1:end">
    <bmlt:trajectory type="linear" initialValue="0" targetValue="100"/>
  </bmlt:parametervaluechange>
</bml>
```

6.6.8.3 The Interrupt Shorthand

The interrupt attribute is a shorthand for the SAIBA Behavior Planner to remove a selected set of BML blocks from the multimodal behavior plan before scheduling the content of the BML block it is in. BML Example 57 illustrates how this shorthand is used, and what SAIBA Behavior Planner functionality it represents.

BML Example 22 Interrupt all behaviors in `bml1`, `bml2`, .., `bmln` before scheduling `bmlNew`. Top: the BML^T shorthand; bottom: an outline of the SAIBA Behavior Planner functionality it represents.

(a) Interrupt shorthand.

```
<bml id="bmlNew" bmlt:interrupt="bml1,bml2,..,bmln">
  bmlNew content
</bml>
```

(b) SAIBA Behavior Planner functionality implemented by the interrupt shorthand.

1. Send a BML block to the Realizer that interrupts `bml1..bmln`:

```
<bml id="bmlInterrupt">
    <bmlt:interrupt id="interrupt1" target="bml1"/>
    <bmlt:interrupt id="interrupt2" target="bml2"/>
    ..
    <bmlt:interrupt id="interruptn" target="bmln"/>
</bml>
```

2. Wait for block end feedback of bmlInterrupt (to make sure `bml1..bmln` are properly removed from the multimodal behavior plan).
3. Send the new BML block bmlNew:

```
<bml id="bmlNew">
    bmlNew content
</bml>
```

6.6.8.4 The onStart Shorthand

The onStart attribute is a shorthand for the SAIBA Behavior Planner to activate a selected set of BML blocks in the multimodal behavior plan whenever a certain BML block starts. BML Example 23 illustrates how this shorthand is used, and what SAIBA Behavior Planner functionality it represents.

BML Example 23 Activate all behaviors in `bml1`, `bml2`, .., `bmln` when bmlNew is started. Top: the BML^T shorthand; bottom: an BML script represented by the shorthand.

(a) Activation shorthand.

```
<bml id="bmlNew" bmlt:onStart="bml1,bml2,..,bmln">
  bmlNew content
</bml>
```

(b) SAIBA Behavior Planner functionality implemented by the onStart shorthand.

```
<bml id="bmlActivate">
    <bmlt:activate id="activate1" target="bml1"/>
    <bmlt:activate id="activate2" target="bml2"/>
    ..
    <bmlt:activate id="activaten" target="bmln"/>
    bmlNew content
</bml>
```

6.7 Discussion

I have introduced BML^T, a BML extension that (amongst other things) allows a formal specification of multimodal behavior generation mechanisms that satisfy the requirements of behavior generation for continuous interaction.

Some of the individual specification mechanisms I have introduced have already appeared in some form as BML extensions, or are used in other multimodal specification languages. This shows their usefulness, sometimes even outside the domain of continuous interaction. The *combination* of the specification mechanisms provided by BML^T provides interesting opportunities for virtual human interaction systems. The prime example of this is the combination of interruption (also available in a simpler form in SmartBody [280]) and pre-planning (also available in the SAL system [258]) that allows graceful interruption with an instantly activated continuation.

BML can be seen as a language that employs a *constraint programming* paradigm: time relations between behaviors are stated in the form of constraints. BML's event/wait system assumes some event-based interaction with the outside world, but this feature is currently underspecified. BML^T extends BML with elements of *meta programming* and *reactive programming*.

Metaprogramming allows the writing of programs that modify other programs (or themselves). BML^T allows meta programming through its activate and interrupt behaviors. These behaviors allow the specification of modifications on the multimodal behavior plan, which are to be applied during their execution.

Reactive programming allows one to specify data flows that automatically propagate change. A spreadsheet is an example of reactive programming: the value of a cell can be defined using a formula that relies on the value of other cells (e.g. =SUM(A1:A10)). If the value in a cell changes, the values of other cells that use it

in their formula are automatically updated. In Elckerlyc, alignment to time values predicted by Anticipators introduces a reactive programming element: the multimodal behavior plan is automatically updated to reflect the updated predictions of an Anticipator. In related work on behavior specification, reactive programming is used for the specification of interactive animation [72] or the specification of robot behavior [118].

6.7.1 Consequences for the SAIBA Behavior Planner

To make use of the continuous interaction capabilities provided by BML^{T} (e.g. using behaviors that interrupt or modify the ongoing multimodal behavior plan), a SAIBA Behavior Planner needs to keep track of the BML blocks it has sent to the Realizer and the execution progress made on them. This functionality can be achieved using the standard BML feedback provided by a Realizer.

Additionally, a SAIBA Behavior Planner might need to know if pre-planning on a block has finished in order to activate it (see the scenario in Section 6.5.3), and must have some rough idea of the global start time of a block (or the end time of its predecessor) in order to specify its alignment to sync points managed by an Anticipator. To satisfy these requirements, the Elckerlyc Realizer provides the SAIBA Behavior Planner with feedback on when the scheduling of a BML block is started and when it is done. It also provides the SAIBA Behavior Planner with predictions on when a behavior block will be started and when it will be finished. The following feedback messages are used for this:

1. *Scheduling start feedback* Indicates that the Realizer has started scheduling a BML block. Provides the predicted start time of a the BML block (based on its append targets and the current time).

2. *Scheduling finished feedback* Indicates that the Realizer has finished scheduling a BML block. Provides the predicted start time (based on its append targets and the current time) and the predicted end time (based on the duration of the scheduled behaviors in the block) of the BML block.

3. *Performance start feedback* In addition to the required information, Elckerlyc provides the start time of the BML block and its predicted end time (based on the duration of the scheduled behaviors in the block).

4. For each sync point in each behavior in the BML block: *Sync-Point Progress Feedback*. In addition to the required information, Elckerlyc provides the local (as offset from the BML block start) and global time of the sync point.

5. *Performance stop feedback* In addition to the required information, this provides the end time of the block.

Note that this feedback provides the SAIBA Behavior planner with incremental predictions (with accuracy increasing with each increment) of the global start and end time of the BML blocks it has sent to the Realizer.

6.7.2 Continuous Interaction Requirements for the Realizer

In a Realizer that supports continuous interaction, the execution of behaviors should not be ballistic, but is instead a continuous process that offers rapid and fine-grained interruption and allows one to get and set parameter values on behaviors during their execution.

Gracious interruptions require behavior implementations that allow alternative continuation (for example: to make the hands return to a rest pose after an interrupted gesture).

BML^{T} provides the specification of the following changes to an ongoing multimodal behavior plan:

1. Activation of pre-planned behavior.
2. Interruption of running or scheduled behavior.
3. Micro-adjustments in the timing of specific behaviors.
4. Setting parameter values on a selected behaviors on the fly.

These changes require a flexible multimodal plan representation that retains information on the behavior specification (as specified in the BML blocks sent to the Realizer). Behaviors in the multimodal behavior plan are identified through their BML block id and behavior id. This allows the Realizer to access the behavior plan and modify parameter values in specific behaviors, interrupt specific behaviors or adjust the timing of specific behaviors. All these modifications to the multimodal behavior plan should occur without violating the time constraints specified for it (through BML). The next chapter shows how such a flexible plan is constructed and maintained in Elckerlyc.

Chapter 7

Scheduling and Multimodal Plan Representation†

A BML Realizer is responsible for executing the behaviors specified in the BML blocks sent to it, in such a way that the time constraints specified in the BML block are satisfied. Realizer implementations, including Elckerlyc, handle this by separating the BML scheduling process from the execution process. The scheduling process is responsible for creating a multimodal behavior plan that is in a suitable form for execution.

Scheduling thus adds new behaviors (specified within a BML block) to the current multimodal behavior plan (Figure 7.1).[1]

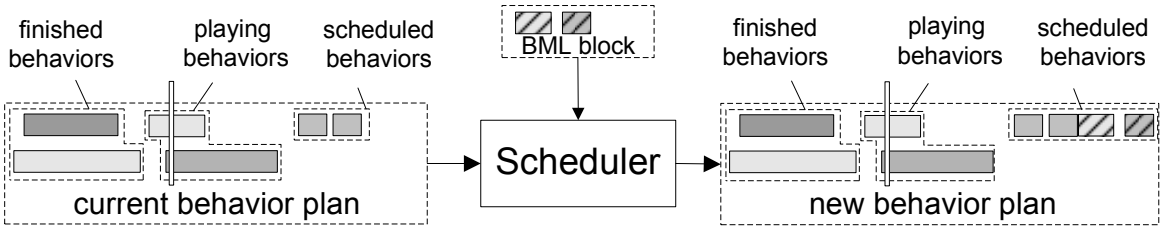


Figure 7.1: The scheduling process

The BML block contains a set of behaviors and the timing constraints that are to be satisfied between the behaviors in the block. In addition to the constraints that are explicitly defined in the BML block, the Scheduler needs to take into account certain implicit constraints that should hold for each BML block. For example, it needs to ensure that for each scheduled behavior, the sync points remain in order (e.g. start before ready, ready before stroke_start, etc). Realizers can impose additional constraints upon the scheduling, for example motivated by biological properties of

†A condensed version of this chapter has been published as:
D. Reidsma, H. van Welbergen, and J. Zwiers. Multimodal Plan Representation for adaptable BML Scheduling, In Proceedings of the 11th International Conference on Intelligent Virtual Agents, 2011, To appear.

[1]The multimodal behavior plan is an abstract representation of the Motor Plan discussed Chapter 4. A behavior might be represented by multiple PlanUnits in the Motor Plan.

humans. Block Level Constraints, as defined by the scheduling attribute in the BML block, define the time constraint between the start of the to be scheduled BML block and the behaviors already present in the current behavior plan.

The BML scheduling problem can be phrased as a constraint programming problem. Both standard constraint programming techniques and custom BML scheduling algorithms have been devised to solve it.

In most BML Realizers, scheduling the stream of BML blocks results in a *rigid* multimodal realization plan in which the timing of all behaviors is fixed. In Elckerlyc however, the multimodal behavior plan is modified at execution time by the execution of interrupt behaviors, by the activation of pre-planned BML blocks and by timing updates from Anticipators. Such modifications should not invalidate the constraints mentioned above. Elckerlyc contributes a *flexible* multimodal plan representation that allows plan modification, while retaining its constraints.

Section 7.1 gives an overview of constraints that act upon a multimodal behavior plan constructed by a BML Scheduler. Section 7.2 shows how the BML Scheduling problem is phrased as a constraint optimization problem and discusses the scheduling solutions used in other Realizers. Section 7.3 discusses Elckerlyc's flexible multimodal realization plan, the construction of this plan and Elckerlyc's mechanisms to keep the scheduling constraints intact under behavior plan changes. Section 7.4 discusses several open issues in scheduling BML and gives an overview of extensions that can enhance Elckerlyc's scheduling process.

7.1 Constraint Specification

The Scheduler adds a BML block represented by behaviors $\mathbf{b}$, their sync points $\mathbf{s}$ and the constraints on these sync points $\mathbf{c}$ to a behavior plan. After scheduling, all default and custom sync points of each behavior in the BML block have been assigned a time. The resulting behavior plan satisfies both the explicit constraints that are expressed in BML, certain implicit constraints that hold for each behavior, a BML block constraint on the start time of the block and typically certain Realizer specific constraints.

7.1.1 Explicit Constraints

Within a BML block, explicit time constraints are expressed between *sync references*. A sync reference consists of either an offset from the start of the BML block, or a pair $[s, o]$, where s is a sync point, defined by the pair $[b, \text{sync id}]$ and o is a time offset (in seconds) from the time of the sync id.[2] b is defined as $[\text{block id}, \text{behavior id}]$. BML Example 24 gives an example of a constraint expressed in a simple BML block.

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 refs $\mathbf{b}$.

[2]For the clarity of this discussion I omit stroke ids here, they can be expressed in a similar manner.

BML Example 24 Behavior `g1` in BML block `bml1` in this example defines a constraint between the sync refs $[[[\text{bml1}, \text{g1}], \text{ready}], 0]$ and $[[[\text{bml1}, \text{beh1}], \text{start}], 1]$.

```
<bml id="bml1">
  <gaze id="g1" ready="beh1:start+1"/>
  ...
</bml>
```

- *at*: sync refs **a** occur at the same time.

For ease of specification and without loss of generality, I define each time constraint as acting between two sync refs. A constraint is an absolute constraint if one of the sync refs is an offset from the start of the BML block. A constraint is a relative constraint if both sync refs are triples of behavior id, sync id and offset time.

An absolute 'at' constraint c_a on a sync point with id s in behavior b at offset o from the start of the BML block is defined by

$$c_a = [[b, s], o] \tag{7.1}$$

Absolute before and after constraints c_{a_b} and c_{a_a} on a sync point with id s in behavior b at offset o from the start of the BML block are defined as

$$c_{a_b} = [[b, s], o] \tag{7.2}$$

$$c_{a_a} = [[b, s], o] \tag{7.3}$$

A relative 'at' constraint c_r between sync refs $[[b_1, s_1], o_1]$ and $[[b_2, s_2], o_2]$ is defined by

$$c_r = [[b_1, s_1], [b_2, s_2], o_2 - o_1] \tag{7.4}$$

Relative before c_{r_b} and relative after c_{r_a} constraints between sync refs $[[b_1, s_1], o_1]$ and $[[b_2, s_2], o_2]$ are defined as follows:

$$c_{r_b} = [[b_1, s_1], [b_2, s_2], o_2 - o_1] \tag{7.5}$$

$$c_{r_a} = [[b_1, s_1], [b_2, s_2], o_2 - o_1] \tag{7.6}$$

A relative before constraint $[[b_1, s_1], [b_2, s_2], o]$ can be converted to relative after constraint c_{r_a} using

$$c_{r_a} = [[b_2, s_2], [b_1, s_1], -o] \tag{7.7}$$

A BML block contains a set of behaviors $\mathbf{b}$, a set of sync points (pairs of behavior id and sync id) $\mathbf{s}$, a set of absolute constraints $\mathbf{c_a}$, a set of absolute before constraints $\mathbf{c_{a_b}}$, a set of absolute after constraints $\mathbf{c_{a_a}}$, a set of relative constraints $\mathbf{c_r}$ and a set of relative after constraints $\mathbf{c_{r_a}}$.[3]

[3]To specify the explicit constraints in a BML block in a unique manner, all relative before constraints are converted to after constraints using equation 7.7.

The function $f : \mathbf{s} \rightarrow \mathbf{t}$ maps a sync point s to global time t. The goal of scheduling is to find such a mapping for all sync points in all behaviors in the block in such a way that all constraints are satisfied. The function $\mathrm{blockstart} : \mathbf{u} \rightarrow \mathbf{t}$ maps the block id u to its global start time t. In Section 7.1.5 I show how the $\mathrm{blockstart}$ is defined, given the scheduling attribute of the BML block. The BML block defines the following explicit constraints on f:

$$\forall[[[\text{bmlid}, \text{behid}], s], o] \in \mathbf{c_a}.\, f([\text{bmlid}, \text{behid}], s) = o + \mathrm{blockstart}(\text{bmlid}) \tag{7.8}$$

$$\forall[[[\text{bmlid}, \text{behid}], s], o] \in \mathbf{c_{a_a}}.\, f([\text{bmlid}, \text{behid}], s) \geq o + \mathrm{blockstart}(\text{bmlid}) \tag{7.9}$$

$$\forall[[[\text{bmlid}, \text{behid}], s], o] \in \mathbf{c_{a_b}}.\, f([\text{bmlid}, \text{behid}], s) \leq o + \mathrm{blockstart}(\text{bmlid}) \tag{7.10}$$

$$\forall[[b_1, s_1], [b_2, s_2], o] \in \mathbf{c_r}.\, f(b_1, s_1) + o = f(b_2, s_2) \tag{7.11}$$

$$\forall[[b_1, s_1], [b_2, s_2], o] \in \mathbf{c_{r_a}}.\, f(b_1, s_1) + o \geq f(b_2, s_2) \tag{7.12}$$

7.1.2 Implicit Constraints

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

1. Sync points may not occur before the block they are in is started (equation 7.13).
2. Behaviors should have a nonzero duration (equation 7.14).[4]
3. The default BML sync points of each behavior must stay in order (equation 7.15).

$$\forall[[\text{bmlid}, \text{behid}], s] \in \mathbf{s}.\, f([\text{bmlid}, \text{behid}], s]) \geq \mathrm{blockstart}(\text{bmlid}) \tag{7.13}$$

$$\forall b \in \mathbf{b}.\, f([b, end]) > f([b, start]) \tag{7.14}$$

$$\begin{aligned} \forall b \in \mathbf{b}. & f([b, start]) \geq f([b, ready) \geq \\ & f([b, strokestart]) \geq f([b, stroke]) \geq \\ & f([b, strokeend]) \geq f([b, relax]) \geq f([b, end]) \end{aligned} \tag{7.15}$$

[4] For a persistent behavior b $f([b, \text{end}]) = \infty$, so b has infinite duration.

7.1.3 Cluster Constraints

A BML block may contain several clusters of behaviors. Each cluster contains a set of behavior connected with 'at' constraints. I define the start of the cluster as the start of the first behavior in the cluster. A cluster can be *grounded*[5], that is, connected to the start of a BML block with an absolute 'at' constraint, or ungrounded. A simple example of an ungrounded cluster is given in BML Example 25.

BML Example 25 A BML script with one ungrounded cluster.

```
<bml id="bml1">
  <speech id="s1"><text>Hello <sync id="world"/>world</text></speech>
  <gesture id="g1" stroke="speech1:world" type="BEAT"/>
</bml>
```

Here the ungrounded cluster contains two behaviors: $[\text{bml1}, \text{s1}]$ and $[\text{bml1}, \text{g1}]$. The constraints defined so far do not determine when an ungrounded cluster starts, other than that it starts after or at the start of the BML block it is in.

A scheduler has the freedom to set up the internal timing of each behavior as it likes, as long as the implicit and explicit constraints defined in the sections above are satisfied. This freedom is typically used to set up the timing of behaviors in such away that the resulting motor behavior is natural. One would like to schedule ungrounded clusters in such a way that gaps between clusters, or, between clusters and the start of the block are minimized, so that they start 'as soon as possible', while retaining this scheduling freedom.

The cluster constraint achieves this by setting up the constraint as one that acts between clusters, without requiring changes to the relative timing of the behavior *within* a cluster. A cluster constraint thus states a synchronization between *one* sync point in an ungrounded cluster and either a sync point in another cluster or the start of its BML block. Thus, in the block bml1 defined above, the cluster constraint would require that either $[\text{bml1}, \text{s1}]$ or $[\text{bml1}, \text{g1}]$ start at $\text{blockstart}(\text{bml1})$. A scheduler has the freedom to select between those solutions. A typical scheduler will pick the latter solution if $[\text{bml1}, \text{g1}]$ is a complex gesture (with the start to stroke taking longer than uttering the word "Hello") and the first solution if it is not.

An ungrounded cluster may contain relative or absolute 'after' constraints. If the gaps between clusters are to be minimized using only one constraint per cluster, this means that the cluster should start at the start of the BML block it is in, or that one of its 'after' constraints is satisfied as an 'at' constraint. If an ungrounded cluster has no 'after' constraints, then it should start at the start of the BML block it is in.

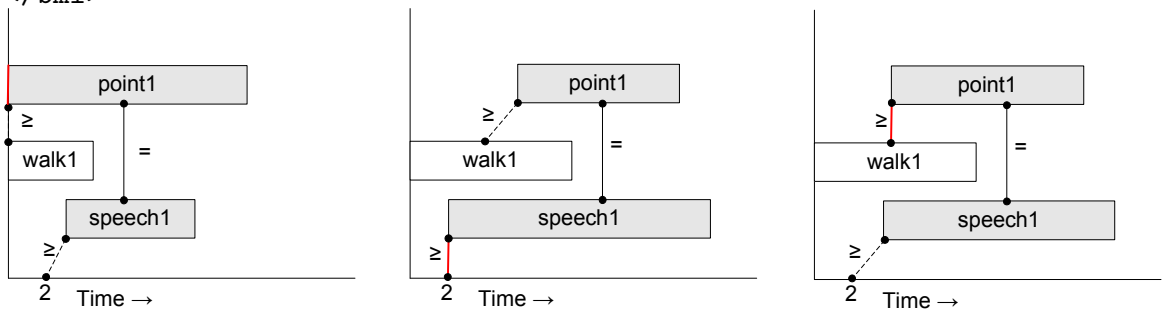
BML Example 26 shows an example of a BML script with both absolute and relative after constraints. In this script, the virtual human walks to a painting and points out a certain spot on the painting. The pointing should start after the virtual human has reached its location. The stroke of the pointing gesture is synchronized to the speech. The speech is constrained to start at least 2 seconds after

[5]The notion of grounding was taken from [144].

the start of `bml1`. BML Example 26 has 2 clusters, one contains behaviors `speech1` and `point1`, and one contains only behavior `walk1`. Both clusters are ungrounded. The cluster containing only `walk1` does not have any after constraints, so `walk1` starts at $\mathrm{blockstart}(\mathrm{bml1})$. The cluster containing `speech1` and `point1` has two after constraints. To satisfy the cluster constraint, either $[\mathrm{bml1}, \mathrm{point1}]$ should start at $\mathrm{blockstart}(\mathrm{bml1})$[6], or one of the after constraints should be satisfied as an at constraint, that is: $[\mathrm{bml1}, \mathrm{speech1}]$ should start at $\mathrm{blockstart}(\mathrm{bml1}) + 2$ or $[\mathrm{bml1}, \mathrm{point1}]$ should start at $f([[\mathrm{bml1}, \mathrm{walk1}], relax])$.

BML Example 26 Cluster constraint example with relative and absolute after constraints. The different options for the cluster constraint are shown in red.

```
<bml id="bml1">
    <speech id="speech1">
        <text>
        As you can see on <sync id="s1"> this painting,
        </text>
    </speech>
    <gesture id="point1" type="POINT" target="painting1_point1"
      stroke="speech1:s1"/>
    <locomotion id="walk1" target="painting1"/>
    <constraint id="c1">
        <after ref="walk1:relax">
            <sync ref="point1:start"/>
        </after>
        <after ref="2">
            <sync ref="speech:start"/>
        </after>
    </constraint>
</bml>
```



[6]This solution implicitly requires that the relax of the `walk1` is at $\mathrm{blockstart}(\mathrm{bml1})$, that is, that the virtual human is already at the desired location. Note that starting `speech1` at $\mathrm{blockstart}(\mathrm{bml1})$ is not a solution, since the absolute after constraint requires that the start of `speech1` is at $\mathrm{blockstart}(\mathrm{bml1}) + 2$ or later.

7.1.3.1 Cluster Properties

To define the cluster constraint formally, I first introduce some predicates that indicate the cluster properties of a behavior.

The predicate $\mathrm{DirectLink}(b_1, b_2)$ expresses that two behaviors b_1 and b_2 are directly connected by an 'at' constraint.

$$\begin{aligned}\mathrm{DirectLink}(b_1, b_2) \equiv \exists o, s_1, s_2.\, ([[b_1, s_1], [b_2, s_2], o] \in \mathbf{c_r} \vee \\ [[b_2, s_2], [b_1, s_1], o] \in \mathbf{c_r})\end{aligned} \tag{7.16}$$

The predicate $\mathrm{IsConnected}(c, d)$ expresses that two behaviors c and d are connected by a chain of 'at' constraints.

$$\begin{aligned}\mathrm{IsConnected}(c, d) \equiv \\ \exists N > 0.\, \forall i \in 0..N-1.\, b_i \in \mathbf{b} \wedge \mathrm{DirectLink}(b_i, b_{i+i}) \wedge c = b_0 \wedge d = b_N\end{aligned} \tag{7.17}$$

The predicate $\mathrm{DirectGround}$ expresses that a behavior b has an absolute constraint.

$$\mathrm{DirectGround}(b) \equiv \exists o, s.\, [[b, s], o] \in \mathbf{c_a} \tag{7.18}$$

The predicate $\mathrm{DirectAfterGround}$ expresses that a behavior b has an absolute after constraint.

$$\mathrm{DirectAfterGround}(b) \equiv \exists o, s.\, [[b, s], o] \in \mathbf{c_{a_a}} \tag{7.19}$$

The predicate $\mathrm{IsGrounded(b)}$ expresses that a behavior b is part of a grounded cluster of behaviors.

$$\begin{aligned}\mathrm{IsGrounded}(b) \equiv \\ \mathrm{DirectGround}(b) \vee \exists c.\, (\mathrm{IsConnected}(b, c) \wedge \mathrm{DirectGround}(c))\end{aligned} \tag{7.20}$$

The predicate $\mathrm{OnBlockStart}([\mathsf{bmlid}, \mathsf{behid}])$ expresses that the cluster of behavior $[\mathsf{bmlid}, \mathsf{behid}]$ starts at $\mathrm{blockstart}(\mathsf{bml1})$.

$$\begin{aligned}\mathrm{OnBlockStart}([\mathsf{bmlid}, \mathsf{behid}]) \equiv \\ f([[\mathsf{bmlid}, \mathsf{behid}], start]) = \mathrm{blockstart}(\mathsf{bmlid}) \vee \\ (\exists c \in \mathbf{b}.\, \mathrm{IsConnected}([\mathsf{bmlid}, \mathsf{behid}], c) \wedge f([c, start]) = \mathrm{blockstart}(\mathsf{bmlid}))\end{aligned} \tag{7.21}$$

The predicate $\mathrm{OnAbsAfterConstraint(b)}$ expresses that the cluster of behavior b satisfies one of its absolute after constraints as an at constraint.

$$\begin{aligned}\mathrm{OnAbsAfterConstraint}([\mathsf{bmlid}, \mathsf{behid}]) \equiv \\ \exists [[b_1, s_1], o] \in \mathbf{c_{a_a}}.\, (([\mathsf{bmlid}, \mathsf{behid}] = b_1 \vee \\ \mathrm{IsConnected}([\mathsf{bmlid}, \mathsf{behid}], b_1)) \wedge \\ f([b_1, s_1]) + o = \mathrm{blockstart}(\mathsf{bmlid}))\end{aligned} \tag{7.22}$$

The predicate $\mathrm{OnRelAfterConstraint(b)}$ expresses that the cluster of behavior b satisfies one of its relative after constraints as an at constraint.

$$\begin{aligned}\mathrm{OnRelAfterConstraint}([\mathsf{bmlid}, \mathsf{behid}]) \equiv \\ \exists [[b_1, s_1], [b_2, s_2], o] \in \mathbf{c_{r_a}}.\, (([\mathsf{bmlid}, \mathsf{behid}] = b_1 \vee \\ \mathrm{IsConnected}([\mathsf{bmlid}, \mathsf{behid}], b_1)) \wedge \\ f(b_1, s_1) = f(b_2, s_2) + o)\end{aligned} \tag{7.23}$$

7.1.3.2 The Cluster Constraint

Using the cluster properties defined above, the cluster constraint is defined as:

$$\begin{aligned} \neg\,\mathrm{IsGrounded}([\mathsf{bmlid}, \mathsf{behid}]) \rightarrow \\ \mathrm{OnBlockStart}([\mathsf{bmlid}, \mathsf{behid}]) \vee \\ \mathrm{OnAbsAfterConstraint}([\mathsf{bmlid}, \mathsf{behid}]) \vee \\ \mathrm{OnRelAfterConstraint}([\mathsf{bmlid}, \mathsf{behid}]) \end{aligned} \tag{7.24}$$

7.1.4 Realizer Specific Constraints

Realizers might impose additional custom 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 biologically motivated. A Realizer might, for example, forbid solutions that require a virtual human to gesture at speeds beyond its physical ability.

7.1.5 Block Level Constraints

The scheduling attribute defined in the BML Block defines constraints on the start of the block in relation to the set of current behaviors in the multimodal behavior plan $\mathbf{B}$ and the current global time ct. Core BML defines the following scheduling attributes:

1. `merge`: start the block at ct (equation 7.25).
2. `replace`: completely replaces the current behavior, start the block at ct (equation 7.26).
3. `append`: start the block directly after all behaviors in the current plan are finished (equation 7.27).

$$\mathrm{schedulingattribute}(\mathsf{bml1}) = \mathsf{merge} \rightarrow \mathrm{blockstart}(\mathsf{bml1}) = \mathsf{ct} \tag{7.25}$$

$$\mathrm{schedulingattribute}(\mathsf{bml1}) = \mathsf{replace} \rightarrow \mathrm{blockstart}(\mathsf{bml1}) = \mathsf{ct} \tag{7.26}$$

$$\begin{aligned} \mathrm{schedulingattribute}(\mathsf{bml1}) = \mathsf{append} \rightarrow \mathrm{blockstart}(\mathsf{bml1}) \geq \mathsf{ct} \wedge \\ \forall b \in \mathbf{B}.\, f(b, end) \leq \mathrm{blockstart}(\mathsf{bml1}) \wedge \\ ((\exists b \in \mathbf{B}.\, f(b, end) = \mathrm{blockstart}(\mathsf{bml1})) \vee (\mathrm{blockstart}(\mathsf{bml1}) = \mathsf{ct})) \end{aligned} \tag{7.27}$$

7.1.6 Additional Requirements of Scheduling

- *Real-time*: to ensure that BML can be sent to a Realizer in a continuous stream, scheduling behaviors should be (preferably several times) faster than executing the behaviors.[7]

- *Invariant to behavior ordering*: for authoring consistency, the execution of behaviors should be the same, no matter their ordering within the BML block.

7.2 BML Scheduling Solutions

BML scheduling is the process of adding a BML block to the multimodal behavior plan. This requires finding the timing of all sync points in all behaviors in the BML block that is scheduled, subject to the constraints laid out in the previous section.

The scheduling problem can be phrased as a classic constraint optimization problem. Several BML Realizers have implemented schedulers to solve this problem (or a subset of it, e.g., only the 'at' constraints). In this section, I discuss the generic scheduling problem and the scheduling solutions of the SmartBody[280] and EMBR [111] BML Realizers.[8]

7.2.1 BML Scheduling as a Constraint Optimization Problem

If cluster constraints are disregarded, the BML scheduling problem can be cast into a classic constraint optimization problem:

$$\text{minimize} \quad g(\mathbf{t}) \quad \text{subject to} \quad C_i(\mathbf{t}) \leq c_i, C_j(\mathbf{t}) = c_j, i = 1..n, j = n..m \qquad (7.28)$$

Where $\mathbf{t}$ is a vector containing time values for synchronization points in all behaviors. $C_i(\mathbf{t}) \leq c_i$ encodes an after or before constraint (for example: $t_1 - t_2 \leq 0$ encodes that t_1 should occur before t_2). $C_j(\mathbf{t}) = c_j$ encodes an equality constraint (for example: $t_1 - t_2 = 0$ encodes that t_1 should at the same time as t_2). The implicit constraint defined in equations 7.13 and 7.15 are of a similar form and can also be encoded in $\mathbf{C} = [C_1..C_m]$.

Behavior specific constraints can be added (for example to enforce a minimum duration of a behavior). $g(\mathbf{t})$ defines the cost function of timing $\mathbf{t}$. The cost function could be used to select the 'cheapest' (in terms of behavior stretching/skewing, etc). It could also be used to set up soft constraints.

The constraint optimization problem can be solved for finite domain values of the elements of $\mathbf{t}$ (e.g. integer values, for example with a resolution of 10ms) using constraint programming techniques. Several toolkits implement such techniques, for example JaCop[9] or Prolog.

[7] This constraint depends not only on the scheduling algorithm, but also on the speed of the scheduling hardware.

[8] The authors of other Realizers do not provide public documentation of their scheduling solutions.

[9] `http://jacop.osolpro.com/`

Alternatively, a solution to the optimization problem can be numerically *approximated*, for example using standard numerical nonlinear constraint optimization methods [88], for which several toolkits are available, such as the Open Optimization Library[10], Matlab's optimization library [188] or SNOPT [274].

If cluster constraints are not taken into account, there is are an infinite number of equally valid values for start time of each ungrounded cluster. There does not seem to be a trivial way to cast the cluster constraints into the simple constraint form required for a constraint optimization solver. However, since cluster constraints do not modify time constraints within clusters, the constraint optimization problem can be solved for each cluster separately and a simple algorithm can be used to combine the clusters. Alternatively, the problem can be cast into a more generic constraint programming problem. Such constraint programming problems can also be solved by the toolkits mentioned above.

Constraint programming forms a nice conceptual framework for describing the BML scheduling problem. However, using constraint solvers to solve the problem might be computationally expensive. Furthermore, defining a cost function for the naturalness of the resulting schedule is a challenging problem. Current schedulers (including Elckerlyc) therefore apply some simplifications to the problem, trading scheduling naturalness and expressivity (that is: they might disallow the scheduling of certain valid BML blocks) for ease-of-implementation and scheduling computation time.

7.2.2 SmartBody Scheduling

SmartBody's [280] scheduling algorithm assigns an absolute timing for syncs in each behavior by processing the behaviors in the order in which they occur in the BML block. Each behavior is scheduled to adhere to their BML block timing constraint and 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 uniformly, to avoid discontinuities in animation speed. The scheduling mechanism can result in scheduling some time constraints into the past (that is, before the start of the BML block). A final normalization pass shifts, where needed, clusters of behaviors forward in time to fix this. SmartBody cannot handle before/after constraints yet, but does comply with all explicit constraints, implicit constraints, and cluster constraints that do not concern before and after constraints (constraints 7.8, 7.11,7.13-7.15,7.24).

As the SmartBody scheduling algorithm schedules behaviors in BML order, the resulting schedule may be dependent on the order of the behaviors in a BML block. At worst, BML cannot be scheduled in one behavior order while it can be in another. For example, the BML block in Figure 27b cannot be scheduled because the timing of the `nod1` is determined first, and the scheduler attempts to re-time `speech1` to adhere to this timing. Most speech synthesis systems, including those used in Smart-

[10] `http://ool.sourceforge.net/`

Body disallow such retiming. If the behavior order is changed, as in Figure 27a, then `speech1` is scheduled first, and `nod1` will adhere to the timing imposed by `speech1`. Since a head nod allows flexible retiming, scheduling of this re-ordered BML block will succeed.[11]

BML Example 27 Two BML scripts demonstrating SmartBody's order dependent scheduling solution.

(a) 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>
```

(b) 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>
```

This failure of the SmartBody scheduling algorithm to scheduled behaviors in one behavior order while achieving a schedule in another order is not limited to BML blocks containing speech. BML Example 28 shows another example. The stroke of behavior `beat1` should occur at the same time as the stroke of `nod1`. `nod1` should start at $\mathrm{blockstart}(\mathrm{bml1}) + 3$ and end at $\mathrm{blockstart}(\mathrm{bml1}) + 5$. If `beat1` is scheduled first, it is likely to constrain the stroke of `nod1` in such a way that it is before $\mathrm{blockstart}(\mathrm{bml1}) + 3$, thus violating its start constraint and resulting in a scheduling failure. If however `nod1` is scheduled first, all constraints can be achieved and the BML block is scheduled without error.

The SmartBody scheduling algorithm might also fail when scheduling behaviors that contain a cycle of 'at' constraints in combination with time offsets. BML Example 29 illustrates this. Nod behavior `nod1` is scheduled to start 2 seconds before gesture `g1` and end 3 seconds after gesture `g2`. Gestures `g1` and `g2` occur in sequence. If the gestures are scheduled first, the BML block is scheduled without a problem. However, if the head nod is scheduled first, and its duration is shorter than 5 (see BML Example 29b), then `g1` and `g2` can no longer be scheduled.

Furthermore, SmartBody provides no mechanisms to find the most natural skewing/stretching solution if one is needed; it simply skews behaviors based on their position in the BML block.

[11] To provide some error feedback on this scheduling peculiarity, the SmartBody *parser* enforces BML authors to put all their speech behaviors at the front of the BML block.

BML Example 28 SmartBody's order dependent scheduling solution may fail on behaviors other than speech.

(a) BML block that can be scheduled using the SmartBody scheduling algorithm.

```
<bml id="bml1">
  <head id="nod1" start="3" end="5" action="ROTATION" rotation="NOD"/>
  <gesture id="beat1" stroke="nod1:stroke" type="BEAT" hand="RIGHT"/>
</bml>
```

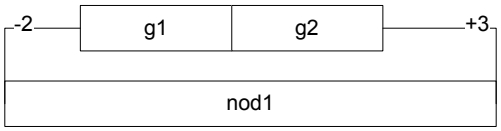
(b) BML block that cannot be scheduled using the SmartBody scheduling algorithm.

```
<bml id="bml1">
  <gesture id="beat1" stroke="nod1:stroke" type="BEAT" hand="RIGHT"/>
  <head id="nod1" start="3" end="5" action="ROTATION" rotation="NOD"/>
</bml>
```

BML Example 29 Failure to schedule a BML block containing a cycle and time offsets using the SmartBody scheduling algorithm.

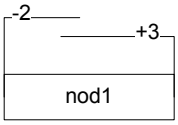
(a) BML block that can be scheduled using the SmartBody scheduling algorithm.

```
<bml id="bml1">
  <gesture id="g1" type="BEAT">
  <gesture id="g2" start="g1:end" type="BEAT">
  <head id="nod1" repeats="5" start="g1:start-2" end="g2:end+3"
    action="ROTATION" rotation="NOD"/>
</bml>
```



(b) BML block that might not be scheduled using the SmartBody scheduling algorithm.

```
<bml id="bml1">
  <head id="nod1" repeats="5" start="g1:start-2" end="g2:end+3"
    action="ROTATION" rotation="NOD"/>
  <gesture id="g1" type="BEAT">
  <gesture id="g2" start="g1:end" type="BEAT">
</bml>
```



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.

It might be possible to solve the ordering issues of the SmartBody scheduling by sorting the behaviors before scheduling them. As illustrated by the examples above, sorting should at least take into account behavior rigidity, placement of absolute constraints and occurrence of dependency loops, so implementing a sorting algorithm is not trivial.

7.2.3 EMBR

EMBR uses a constraint programming technique to solve the scheduling problem (in a similar way as proposed in Section 7.2.1). For ease of implementation it does not implement cluster constraints and instead requires certain properties of the BML blocks it can schedule:

1. The BML block should contain at most one speech behavior.

2. Each connected cluster of behaviors in the BML block is grounded.

The EMBR scheduler first solves the absolute value of all BML sync points in speech. A timing constraint solver then solves the timing of *only* the nonverbal behaviors. Synchronization constraints might require the stretching or skewing of behavior phases as compared to the defaults in a behavior lexicon. The constraint solver uses the sum of 'errors' (in seconds) of the stretch or skew over all behaviors as its cost function. Thus it finds solutions in which the overall stretch/skew is minimized.

Unlike the SmartBody scheduler discussed above, the EMBR scheduler [111, 144] 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).

7.3 Scheduling and Plan Representation in Elckerlyc

BML^{T} allows one to describe the synchronization of BML behaviors to anticipated time events. It adds an append-after scheduling attribute that allows one to append a BML block to a specific set of behaviors (rather than to all behaviors) in the multimodal behavior plan. The additional constraints posed by this on Elckerlyc's behavior plan are described in Section 7.3.1.

Elckerlyc is designed specifically for continuous interaction. Its multimodal behavior plan is continually updated at execution time by Anticipator predictions, interruption of ongoing behavior using interrupt behaviors and activation of new behaviors using Elckerlyc's pre-plan/activation mechanisms. Elckerlyc's flexible plan representation allows these modifications, while keeping the constraints on the behavior plan consistent. Plan changes required by continuous interaction are specified in terms of behaviors and sync points in BML. To allow changes to the Motor

Plan, Elckerlyc maintains information on how it relates to the original BML specification that created it.

Currently Elckerlyc uses a slightly extended version of the SmartBody scheduling algorithm for its scheduling. Elckerlyc is designed in such a way that this scheduling algorithm can easily be changed by an alternative scheduling algorithm at a later stage.

7.3.1 Additional Behavior Plan Constraints In Elckerlyc

7.3.1.1 Anticipator Constraints

Elckerlyc's multimodal behavior plan is designed to allow micro adjustments in its timing. Such time adjustments are steered by Anticipators. An Anticipator instantiates synchronization points that can be used in BML blocks to constrain the timing of behaviors. It uses perceptions of events in the real world to continuously update the timing of its sync points, by extrapolating the perceptions into *predictions* of the timing of future events. An anticipator sync a is defined by $a = [\text{anticipatorid}, \text{syncid}]$.

Constraint c_{ant} describes an 'at' constraint on sync with id s in behavior b at offset o from the anticipator sync a.

$$c_{ant} = [[b, s], o, a] \tag{7.29}$$

A sync point should be connected to at most one anticipator sync with an 'at' constraint.

Constraint c_{ant_a} describes an 'after' constraint on sync with id s in behavior b at offset o from the anticipator sync a.

$$c_{ant_a} = [[b, s], o, a] \tag{7.30}$$

Constraint c_{ant_b} describes a 'before' constraint on sync with id s in behavior b at offset o from the anticipator sync a.

$$c_{ant_b} = [[b, s], o, a] \tag{7.31}$$

In addition to a set of behaviors $\mathbf{b}$, a set of sync points (pairs of behavior id and sync id) $\mathbf{s}$, a set of absolute constraints $\mathbf{c_a}$, a set of absolute before constraints $\mathbf{c_{a_b}}$, a set of absolute after constraints $\mathbf{c_{a_a}}$, a set of relative constraints $\mathbf{c_r}$ and a set of relative after constraints $\mathbf{c_{r_a}}$, a BML$^{\text{T}}$ block contains a set of anticipator syncs $\mathbf{a}$, a set of Anticipator constraints $\mathbf{c_{ant}}$, a set of Anticipator after constraints $\mathbf{c_{ant_a}}$ and a set of Anticipator before constraints $\mathbf{c_{ant_b}}$.

Anticipators provide a global time for their sync points. The function $g : \mathbf{a} \rightarrow \mathbf{t}$ maps an Anticipator sync a to its global time t. The value of $g(a)$ is completely defined by the time prediction of a's Anticipator. Anticipator constraints add the following explicit constraint to the behavior plan:

$$\forall [[b, s], o, a] \in \mathbf{c_{ant}}.\, f([b, s]) + o = g(a) \tag{7.32}$$

$$\forall[[b,s],o,a] \in \mathbf{c_{ant_a}}.\, f([b,s]) + o \geq g(a) \tag{7.33}$$

$$\forall[[b,s],o,a] \in \mathbf{c_{ant_b}}.\, f([b,s]) + o \leq g(a) \tag{7.34}$$

7.3.1.2 Cluster Constraints

Anticipators extend Elckerlyc's notion of 'grounding'. In Elckerlyc, a behavior is grounded not only if it is connected to an absolute 'at' constraint but also if it is connected to an Anticipator sync point. The $\mathrm{DirectGround}$ predicate is updated to reflect this (see equation 7.35).

$$\mathrm{DirectGround}(b) \equiv \exists o,s.\,[[b,s],o] \in \mathbf{c_a} \vee \exists o,s.\,[[b,s],o,a] \in \mathbf{c_{ant}} \tag{7.35}$$

The predicate $\mathrm{OnAbsAfterAntConstraint(b)}$ expresses that the cluster containing behavior b satisfies one of its absolute anticipator 'after' constraints as an 'at' constraint.

$$\begin{aligned}&\mathrm{OnAbsAfterAntConstraint}([\textsf{bmlid},\textsf{behid}]) \equiv\\ &\exists[[b_1,s_1],o,a] \in \mathbf{c_{a_{ant_a}}}.\,((([\textsf{bmlid},\textsf{behid}] = b_1 \vee\\ &\qquad \mathrm{IsConnected}([\textsf{bmlid},\textsf{behid}],b_1)) \wedge\\ &\qquad f([b_1,s_1]) + o = g(a))\end{aligned} \tag{7.36}$$

The updated cluster constraint then becomes:

$$\begin{aligned}&\neg\,\mathrm{IsGrounded}([\textsf{bmlid},\textsf{behid}]) \rightarrow\\ &\mathrm{OnBlockStart}([\textsf{bmlid},\textsf{behid}]) \vee\\ &\mathrm{OnAbsAfterConstraint}([\textsf{bmlid},\textsf{behid}]) \vee\\ &\mathrm{OnRelAfterConstraint}([\textsf{bmlid},\textsf{behid}]) \vee\\ &\mathrm{OnAbsAfterAntConstraint}([\textsf{bmlid},\textsf{behid}])\end{aligned} \tag{7.37}$$

7.3.1.3 Block Level Constraint

In addition to the Core BML `merge` and `append` scheduling attributes, BML$^{\mathrm{T}}$ provides the `append-after(X)` scheduling attribute. Append-after starts a BML block directly after a selected set of behaviors (those from a BML block in $\mathbf{X}$) in the current behavior plan are finished (equation 7.38).

$$\begin{aligned}&\mathrm{schedulingattribute}(\textsf{bml1}) = \textsf{append-after}(\mathbf{X}) \rightarrow\\ &\qquad \mathrm{blockstart}(\textsf{bml1}) \geq \textsf{ct} \wedge\\ &\qquad (\forall[\textsf{bmlid},\textsf{behid}] \in \mathbf{B}.\,\textsf{bmlid} \in \mathbf{X} \rightarrow\\ &\qquad\quad f([\textsf{bmlid},\textsf{behid}],end) \leq \mathrm{blockstart}(\textsf{bml1})) \wedge\\ &\qquad ((\exists[\textsf{bmlid},\textsf{behid}] \in \mathbf{B}.\,f([\textsf{bmlid},\textsf{behid}],end) = \mathrm{blockstart}(\textsf{bml1}) \wedge\\ &\qquad\quad \textsf{bmlid} \in \mathbf{X}) \vee\\ &\qquad\quad (\mathrm{blockstart}(\textsf{bml1}) = \textsf{ct}))\end{aligned} \tag{7.38}$$

7.3.2 Elckerlyc's Plan Representation

Elckerlyc's multimodal behavior plan can be updated continuously at run-time: Anticipator timing updates modify the timing of behaviors synchronized to them, behaviors are removed from the plan or pre-planned behaviors are activated. Anticipator updates and interruption of behavior affects the timing of sync points of behaviors and/or the start time of the BML blocks they are in. To allow these updates the multimodal behavior plan needs to be represented in a flexible manner.

Central to Elckerlyc's plan representation is the Peg Board. The sync points of each behavior in the multimodal plan are associated with a TimePegs on the Peg Board. These TimePegs can be moved, changing the timing of the associated sync point. If two sync points are connected by an 'at' constraint, they share the same TimePeg. This TimePeg can then be moved without violating the 'at' constraint, because this simultaneously changes the actual time of both sync points. TimePegs provide local timing (that is, offset from the start of the block). Each TimePeg is connected to a BMLBlockPeg, that provides a flexible representation of the start time of a BML block. If the BMLBlockPeg is moved, all TimePegs associated with it move along. This allows one to move the block as a whole, keeping the intra-block constraints consistent (see Figure 7.2).

A dedicated BML Block management state machine automatically updates the timing of the BMLBlockPegs in reaction to behavior plan modifications that occur at run-time. It maintains the BML Block constraints described in Section 7.1.5 and Section 7.3.1.3.

7.3.3 Resolving Constraints to TimePegs

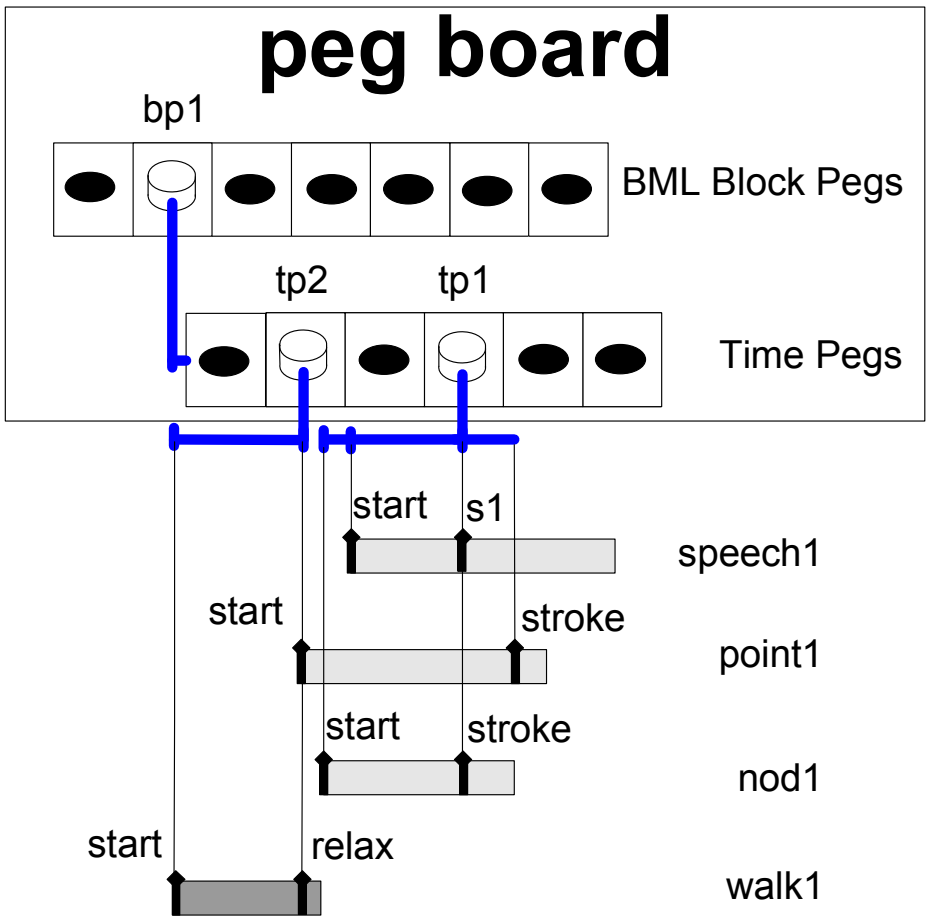
Relative 'at' synchronization constraints that share a sync point (behavior id, sync id pair) should be connected to the same TimePeg. That is: relative 'at' constraints $c_{r_1} = [s_1, s_2, o]$ and $c_{r_2} = [s_3, s_4, p]$ are assigned to the same TimePeg if

$$s_1 = s_3 \vee s_1 = s_4 \vee s_2 = s_3 \vee s_2 = s_4 \tag{7.39}$$

The timing of constraint c_{r_1} is offset by $o - p$ from the timing of c_{r_2}. Such a fixed, nonzero timing offset between relative 'at' constraints is maintained by an OffsetPeg. An OffsetPeg is a TimePeg that is restrained to stay at a fixed offset to its linked TimePeg. If the OffsetPeg is moved, its linked TimePeg moves with it and vice-versa. If the start sync of a behavior is not constrained, an Engine may be asked to resolve the start sync as an OffsetPeg. That is: the start sync of the behavior is linked to the closest TimePeg connected to another sync point within the behavior (see BML Example 30 for some examples of this). If this TimePeg is moved, the start of the behavior is moved with it. If a behavior is completely unconstrained, a new TimePeg is created and connected to its start sync. BML Example 30 shows how TimePegs are resolved from an example BML constraint specification.

Each BML Block has its own associated BML Block Peg, that defines its global start time. Each TimePeg is linked to one BML Block Peg that is used to determine its global time offset. The TimePegs resulting from intra-block constraints specified in a

BML Example 30 Resolving a BML constraint specification to a TimePegs specification. A TimePeg tp1 connects relative 'at' constraints $[[\text{speech1}, \text{s1}], [\text{nod1}, \text{stroke}], 0]$, and $[[\text{speech1}, \text{s1}], [\text{point1}, \text{stroke}], 0.5]$. Another TimePeg tp2 is created for the 'at' constraint $[[\text{point1}, \text{start}], [\text{walk1}, \text{relax}], 0]$. Since the start time of speech1, nod1, and walk1 is not constrained, they are attached to an OffsetPeg linked to the TimePeg that constrains the closest sync in the respective behaviors. The BML Block itself (with id `bml1`) is connected to BMLBlockPeg `bp1`. All TimePegs are connected to this BMLBlockPeg.




```
<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_point1" stroke="speech1:s1+0.5"/>
    <head id="nod1" action="ROTATION" rotation="NOD" stroke="speech1:s1"/>
    <locomotion id="walk1" target="painting1"/>
</bml>
```

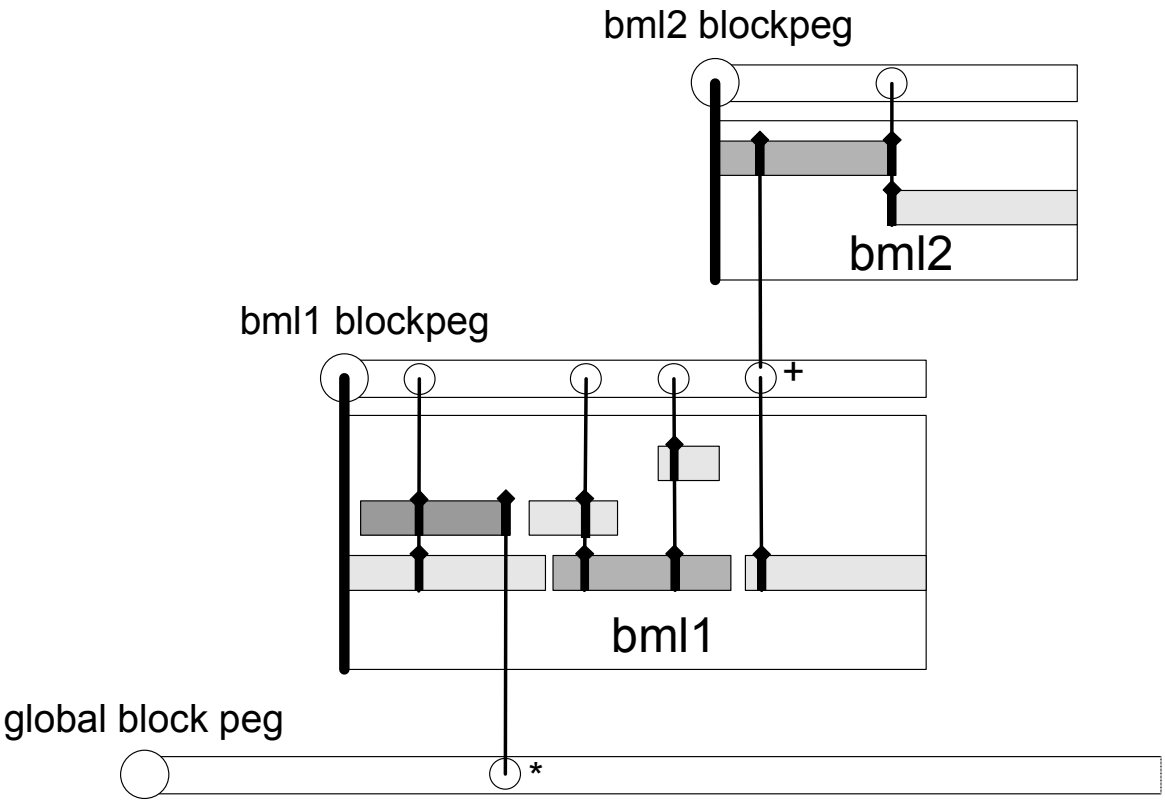


Figure 7.2: Each BML block has its associated BMLBlockPeg. Internal constraints are linked to TimePegs associated with this BMLBlockPeg. BML block `bml1` contains a constraint that is linked to an external TimePeg defined by an Anticipator (marked with *). BML block `bml2` has a constraint whose timing is defined by a TimePeg from BML block `bml1` (marked with +).

BML Block are all linked to the BMLBlockPeg for that specific block. Some behaviors have constraints (and thus TimePegs) that are linked to external BMLBlockPegs, from Anticipators or from other BML Blocks. TimePegs managed by Anticipators are typically hooked up to a special, immovable global BMLBlockPeg which has a 0 global time offset. See Figure 7.2 for a graphical representation of these relations. A first estimate of the time of a BMLBlockPeg is the time at which its scheduling starts. This estimate is used when scheduling the behaviors in the block. When a BML block is actually started (its state is set to IN_EXEC), its BMLBlockPeg time is updated to reflect the actual start time of the block. This does not affect the internal constraints of the block.

7.3.4 Managing BML Block State

A state machine (see Figure 7.3) is used to efficiently maintain the BML block constraints 7.25-7.27 and 7.38. In addition to this, the state machine manages BML^{T}'s SAIBA Behavior Planner shorthand for interrupt (see Chapter 6.6.8.3), the preplanning and activation of BML blocks and the BML and BML^{T} block level feedback.

I explicitly have modeled the state of each BML block. The names of the states

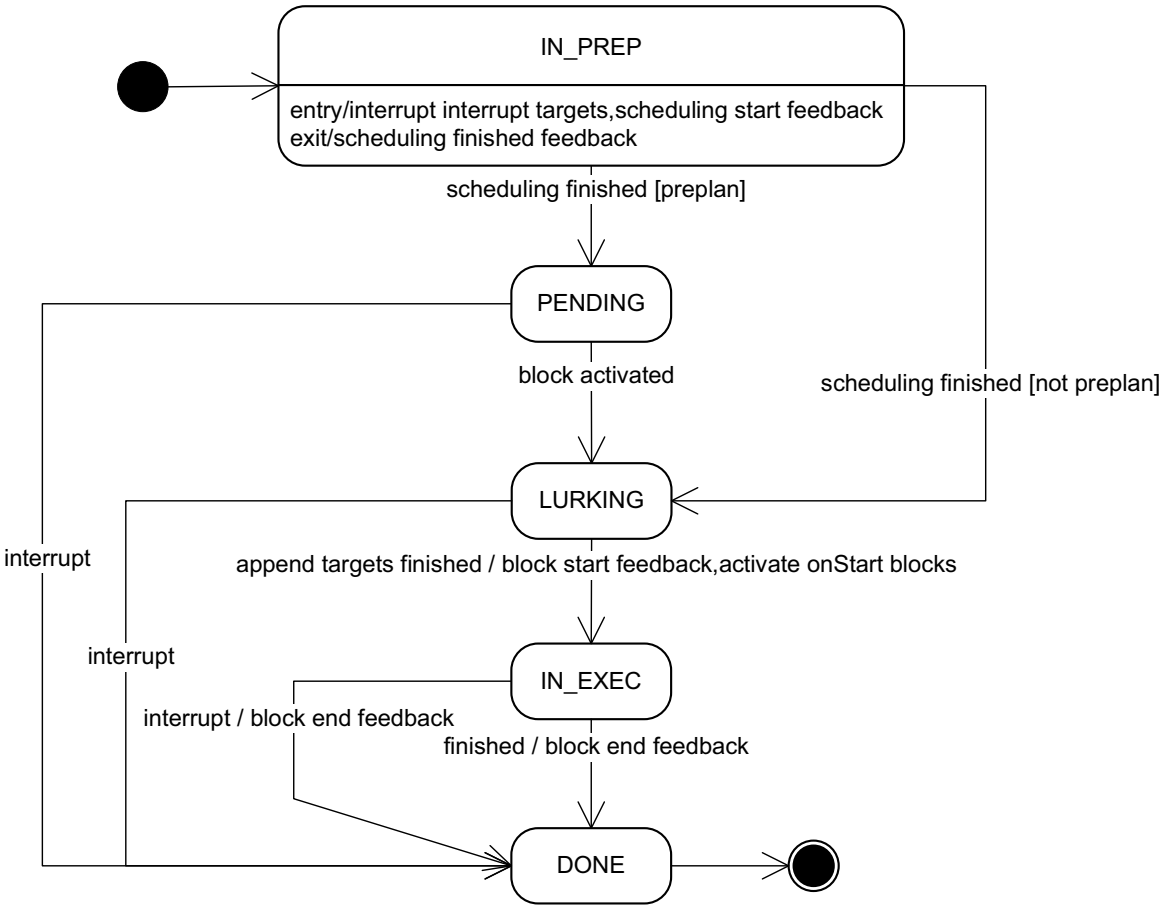


Figure 7.3: BML block state chart. Transitions are labeled with `trigger [guard] / effect`

are inspired by Kopp and Wachsmuth[155]. A BML block starts out in a IN_PREP state, as soon as the Scheduler starts scheduling it. The Scheduler only schedules one BML block at a time, and the blocks are scheduled in order of arrival at the Scheduler. The SAIBA Behavior Planner is notified of the start of the schedule process on a block with an 'start scheduling feedback message'. First the block's interrupt targets are removed from the current behavior plan. If scheduling is finished, a 'scheduling finished feedback message' is send to the SAIBA Behavior Planner. If the block is to be pre-planned, it moves to the PENDING state and awaits activation from subsequent BML blocks or activate behaviors. Otherwise it moves to the LURKING state directly and waits for its append targets (if any) to be finished. Once all append targets are finished, a block start feedback message is sent and the the behaviors within the block are executed. As the block is started, it activates its `onStart` targets and the time of the block's Block Peg is set to the current global time.

In the IN_EXEC state, the state machine uses behavior progress feedback (see Chapter 6.2.4) to keep track of the progress of all behaviors in the BML block it manages. The BML block is finished if end feedback is received for all its behaviors that are in the *current* multimodal behavior plan. If the block finishes, block end feedback is sent and the block moves to DONE state. By matching the BML block ending with the ending of all of its behaviors in the current behavior plan, rather

than with the ending of all the behaviors it originally specified, the ending of the block is found robustly when some of its behaviors are interrupted as it is being executed. This ensures that the append targets of the block are started at their appropriate times; that is: constraints 7.27 and 7.38 are automatically satisfied.

A block can be interrupted using the block interrupt shorthand in another block (see also Chapter 6.6.8.3) at any time while in PENDING, LURKING or IN_EXEC state. An interruption in PENDING or LURKING state simply removes the block from the plan. If the block is interrupted in IN_EXEC state, a block end feedback message is sent before removing it from the plan.

7.3.5 Scheduling in Elckerlyc

In Elckerlyc, scheduling consists of resolving the constraint in a BML block to TimePegs (see Section 7.3.3), and assigning the TimePegs a first prediction of their execution time. Elckerlyc's main scheduling contribution is in its flexible behavior plan representation described in Section 7.3.3 and 7.3.4; Elckerlyc currently uses a simple scheduling algorithm based on that of SmartBody to assign time predictions to the TimePegs. The architecture of Elckerlyc is set up in such a way that this scheduling algorithm can be replaced by alternative and more flexible scheduling algorithms (e.g. that of EMBR, or a another custom constraint solver) at a later stage.

7.3.5.1 Scheduling Architecture

Elckerlyc models scheduling using an interplay between different unimodal Engines that provide the scheduler with detailed information on the 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 TimedPlanUnits, whose timing is linked to TimePegs on the PegBoard. Elckerlyc's Scheduler communicates with these Engines (e.g. Speech Engine, AnimationEngine, see also Figure 7.4) through their abstract interface (see below). It knows which Engine handles each BML behavior type.

Interfacing with the Engines Each Engine is required to implement functionality to:

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

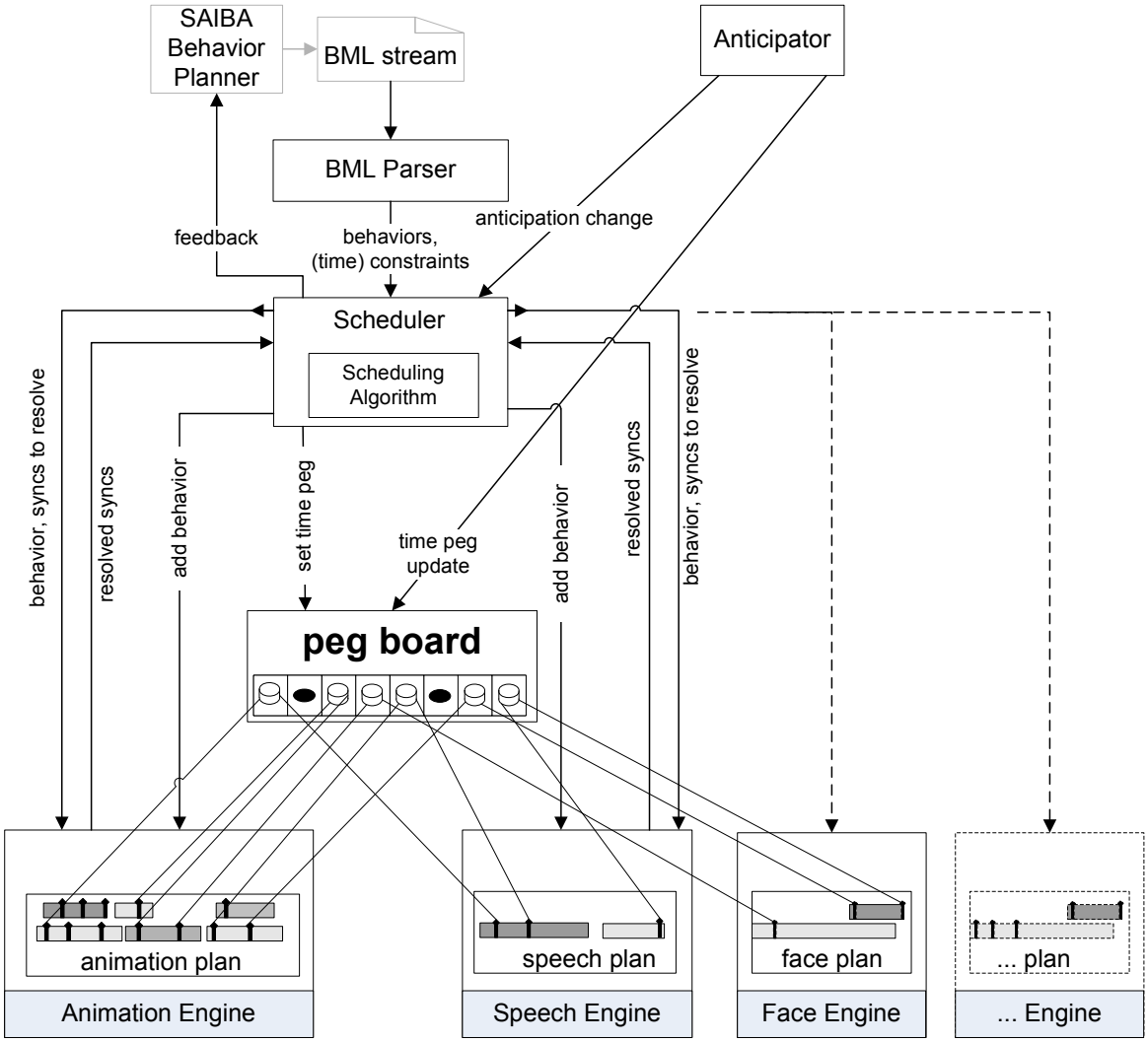


Figure 7.4: Elckerlyc's scheduling architecture

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 TimedPlanUnits. The validity check is typically used to check if a valid plan is retained after an Anticipator moves certain TimePegs. All implemented Engines check if the order of the TimePegs of each behavior is still correct (constraint 7.15). Each Engine can also add validity checks specific to its output modality. These checks are discussed in the section on the specific Engine in Chapter 8.

Scheduling Algorithm The BML block management *process* discussed in Section 7.3.4 delegates the actual scheduling to a scheduling algorithm, using the strategy pattern [83]. The scheduling algorithm assigns a first prediction of the timing of each TimePeg, 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 at a later stage: the BML parsing and block progress management are separated from the scheduling algorithm, and the Engines provide generic interfaces that provide the scheduling algorithm with the timing of unknown constraints on behaviors, given certain known constraints (see Section 7.3.5.1).

Elckerlyc's current scheduling algorithm is based on the SmartBody Scheduler (See [280], Section 7.2.2). The behaviors are sorted first, to avoid some of the ordering issues that occur in SmartBody. The sorting achieves the following order of behavior groups:

1. Rigid behaviors (e.g. synthesized speech) with one or more absolute 'at' constraints or Anticipator 'at' constraints. The timing of these behaviors is completely fixed.

2. All other behaviors that are not completely flexible.

3. Completely flexible behaviors (e.g. `parametervaluechange`), whose timing should adhere to that of other behaviors.

Within groups 1 and 3, the behaviors remain in the order in which they occurred in the BML block. A rigid behavior in group 2 that is connected at two different sync points to a connected set of non-rigid behaviors (that is, it forms a cycle with these behaviors), is ordered to be before these non-rigid behaviors. This ensures that a rigid behavior in a cycle of behaviors is scheduled before its more flexible behaviors. Other than this constraint, the behaviors in group 2 remain in the order in which they occurred in the BML block. This sorting resolves some, but not all of the scheduling order issues of the SmartBody algorithm. It fixes issues similar to BML Example 27b, but not those that might occur in blocks with absolute 'at' constraints (as in BML Example 28b) or cycles and time offsets (as in BML Example 29). By keeping the behaviors within the groups in their original order, it does not introduce any ordering anomalies beyond those that would occur in the original behavior ordering.

The sorted behaviors are processed in this order using the SmartBody scheduling algorithm. The timing of the first behavior in the BML block is constrained only by its absolute time constraints, constraint references to external behaviors and by constraints imposed by Anticipators. 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. Some of these constraints are resolved by already processed behaviors, but others act on subsequent behaviors in the BML block and are yet unknown. Our current scheduler delegates resolving these unknown time constraints directly to the Engine of the behavior it is currently processing. Subsequently, the behavior on which all time constraints are now resolved is added to its Plan.

7.3.6 Managing Adjustments of the Behavior Plan during Behavior Execution

Once a BML block has been scheduled, several changes can occur to its timing at execution time. Such changes are initiated by Anticipators and interrupt behaviors. Elckerlyc's flexible behavior plan representation allows such changes to occur, while keeping the timing constraints on the behaviors intact. 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.

The Anticipator notifies the Scheduler whenever its predictions change, and updates the TimePegs. Since the sync points of behaviors are symbolically linked to the TimePegs, the timing update is handled automatically and satisfies the explicit constraints of Section 7.1.1. Many of such updates are minor and do not require additional changes in the behavior plans. That is: the constraints 7.8-7.12 and 7.13-7.15 remain satisfied. Since the BML Block management state machine dynamically manages the block end, constraints 7.27 and 7.38 are automatically satisfied when an Anticipator update changes the block ending. Elckerlyc currently allows minor timing modification to violate the cluster constraints 7.24.

More significant updates might require re-scheduling of behavior on several modalities. To check if such an update is needed, the Scheduler asks each Engine if its current plan is still valid: it checks if constraints 7.13-7.15 are 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. As an alternative to dropping the behavior, the Scheduler might decide to drop a violated constraint and notify the SAIBA Behavior Planner of this with a warning feedback. The latter is currently not implemented.

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, constraints 7.27 and 7.38 are automatically satisfied, shortening the block whenever this happens.

Currently Elckerlyc does not update the behavior plan to correct violations of the cluster constraints 7.24 that might occur when one or more behaviors are removed

from the behavior plan.

7.4 Discussion

Elckerlyc's flexible behavior plan representation allows one to make microadjustments 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.

Currently, Elckerlyc cannot handle before and after constraints. In Section 7.4.1, I outline the changes that have to be made to Elckerlyc to allow it to handle such constraints while maintaining it flexible plan representation.

Allowing a scheduler to schedule multiple blocks at the same time (in different scheduling threads) improves a Realizer's responsiveness. Section 7.4.2 discusses when two or more BML blocks can be scheduled independently and how this might be implemented in Elckerlyc.

7.4.1 Handling Before and After Constraints

Currently, Elckerlyc cannot handle before and after constraints. To add this functionality, Elckerlyc needs 1) a scheduling algorithm that takes into account before and after constraints and 2) a plan representation that maintains before and after constraints, while allowing micro adjustment in the timing of behaviors.

The scheduling algorithm in EMBR, discussed in Section 7.2.3 is able to handle before and after constraints. This algorithm could be implemented in Elckerlyc as an alternative SchedulingStrategy. In addition to providing before and after constraints, such an implementation would also test and enhance the Elckerlyc's scheduling algorithm independence. To allow the implementation of EMBR's scheduling algorithm in Elckerlyc, the Engine's interface needs to be extended with a function that provides the default timing for a BML behavior. Alternatively and more generally, an Engine could provide the *scheduling cost* of enforcing certain time constraints on the realization of a behavior. A scheduling strategy can then aim to minimize overall scheduling cost. EMBR's scheduling algorithm is then a special case of such a cost minimization strategy. It can be reproduced by providing each engine with cost functions that measure cost as the deviation (in seconds) from the default timing of a behavior.

A relative before or after constraint could be modeled by a pair of linked TimePegs: a BeforePeg and an AfterPeg. If a BeforePeg is moved after its corresponding AfterPeg, the AfterPeg should move with it to retain the before/after constraint. If

the AfterPeg is moved in front of its corresponding BeforePeg, the BeforePeg should move with it. Absolute before and after constraint could be handled by specialized TimePegs that encode the absolute before/after offset and disallow moving TimePegs beyond these offsets.

7.4.2 Multi-threaded Scheduling

Currently, Elckerlyc schedules one BML block at a time, and the blocks are scheduled in order of arrival. This can potentially ruin the rapid interruptibility and adaptability I strive for. The blocks that are to be scheduled form a scheduling queue. If a new BML block is appended to the scheduling queue, it will not be scheduled until the scheduling of the other blocks is finished.

However, it is not always necessary for a Realizer to schedule the BML blocks in order. For example: if a new BML block is queued containing a smile behavior that is to be merged instantly with some ongoing behavior, while pre-planning block that contains some long speech segment is currently being scheduled, the scheduling of the new BML block with the smile should not be delayed until the scheduling of the (unrelated) other block is finished. Formally: scheduling of a BML block can be started as soon as no dependent BML blocks are in front of it in the scheduling queue.
A BML block bmlY is dependent on BML block bmlX if:

1. bmlY activates pre-planned BML block bmlX
2. bmlY is appended after bmlX
3. bmlY interrupts bmlX
4. one or more time constraints in bmlY refer to bmlX
5. one or more behaviors in bmlY refer to bmlX (currently only for interrupt behaviors and parameter change behaviors)

All these properties can be checked by parsing (rather than scheduling) the BML block.

Elckerlyc's scheduler can be extended to a multi threaded Scheduler that spawns new scheduling threads for all independent BML blocks in the queue. Whenever a new BML block is added to the queue, or scheduling of a BML block is finished, the Scheduler will check the current queue and spawn a scheduling thread for all BML blocks that have no more dependencies on other unscheduled blocks.

Chapter 8

Elckerlyc[†]

"Elckerlyc" is a BML Realizer for generating multimodal verbal and nonverbal behavior for virtual humans.[1] A BML Realizer takes a specification of the intended behavior (speech, gaze, gestures, etc.) of a virtual human – written in the Behavior Markup Language (BML) [152], Chapter 6.2 – and executes this behavior through the virtual human.

Elckerlyc builds upon several earlier projects with virtual humans that were carried out in the Human Media Interaction (HMI) lab. During those projects, the need became clear for a number of specific novel characteristics that were not available in the virtual human platforms that we used. Using the experience gained in these earlier projects, Elckerlyc was developed from the ground up as a state-of-the-art BML Realizer.

The main design characteristics of Elckerlyc are that (1) it has been designed specifically for *continuous interaction* with tight coordination between the behavior of a virtual human and its interaction partner; (2) it provides an *adjustable trade-off between the control and naturalness* offered by different animation paradigms (e.g. procedural body animation and physical body animation; MPEG-4 facial animation and morph-based facial animation), allowing the execution of the paradigms simultaneously; and (3) it has been designed to be highly *modular and extensible* and allows adaptations and extensions of the capabilities of the virtual human, without having to make invasive modifications to the Elckerlyc itself. Throughout this Chapter I will demonstrate how these three design goals are achieved.

[†]This chapter is an extended version of the articles:
H. van Welbergen, D. Reidsma, Z.M. Ruttkay and J. Zwiers. Elckerlyc - A BML Realizer for continuous, multimodal interaction with a Virtual Human, Journal on Multimodal User Interfaces, 3(4):271-284, 2010.
D. Reidsma and H. van Welbergen. Elckerlyc in practice – on the integration of a BML Realizer in real applications, Proceedings of the 4th international conference on INtelligent TEchnologies for interactive enterTAINment, 2011, In Press.

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

8.1 Elckerlyc's Predecessors

Several virtual human applications have been developed at the HMI group. Three of these applications have been the most influential on Elckerlyc's design: the Virtual Presenter, the Interactive Virtual Conductor and the Interactive Virtual Dancer.

8.1.1 The Virtual Presenter

Figure 8.1: The Virtual Presenter.

The Virtual Presenter [303, 304] presents in monologues using synchronized speech, pointing gestures, gaze, posture and sheet display (Figure 8.1). The gesture repertoire was later extended with iconic gestures using keyframe animation [142]. Earlier virtual human applications developed at HMI used speech (as generated by a Text-To-Speech synthesizer) to determine the timing on all other modalities. The Virtual Presenter contributed a flexible presentation script that allows authors to define which modality is leading, that is, which modality determines the timing of the other modalities and to change the leading modality over time.

Some of Elckerlyc's global architecture features were inspired by those in the Virtual Presenter. Both the Virtual Presenter and Elckerlyc are platforms that steer a virtual human using a multimodal behavior description. Both systems contain a scheduler that communicates with unimodal planners to construct a multimodal behavior plan. In the Virtual Presenter this plan was a rigid representation of the behavior that is to be executed in a monologue. Elckerlyc contributes a flexible plan representation that allows changes to the plan at run time, as required by continuous

interaction applications. The biologically inspired gaze and pointing MotionUnits developed for the Virtual Presenter (see Chapter 4.2 for implementation details) are reused as custom procedural MotionUnits in Elckerlyc. The Virtual Presenter was set up as a modular and extendible system. However, making use of this extensibility required compile time changes in the Virtual Presenter itself. A source code fork of the Virtual Presenter is still in use in HMI's Virtual Guide [116], a virtual human application that uses a turn-based dialog system for route explanations. It was hard to maintain both systems in parallel, and as a result extensions and changes made for the Virtual Guide were never ported back into the Virtual Presenter. Elckerlyc resolves such maintenance issues by providing extensibility that does not require one to modify Elckerlyc's source, and thus does not require a fork of Elckerlyc to be made for each virtual human application it is used in.

8.1.2 The Interactive Virtual Conductor

Figure 8.2: The Virtual Conductor, Photo: Henk Postma, Stenden Hogeschool

The Virtual Conductor [178, 229] is capable of leading, and reacting to, live musicians in real time (see Figure 8.2). The conductor possesses knowledge of the music to be conducted, and it is able to translate this knowledge into gestures and to produce these gestures. The conductor extracts features through audio processing algorithms as the music is played and reacts to them, based on knowledge of the score. The reactions are tailored to elicit the desired response from the musicians. In addition to indicating the beat, the conductor can provide style information, for example using gestures that indicate that the music should be played louder/softer or that provide an entrance cue.

Clearly, if an ensemble is playing too slowly or too fast, a (human) conductor should lead them back to the correct tempo. He can choose to lead strictly or more leniently, but completely ignoring the musician's tempo and conducting like a metronome set at the right tempo will not work. A conductor must incorporate some sense of the actual tempo at which the musicians play in his conducting, or else he will lose control. If the musicians play too slowly, the virtual conductor will conduct a little bit faster than they are playing. When the musicians follow, it will

conduct faster still, till the correct tempo is reached again. In order to do this, the Virtual Conductor continuously makes a prediction of how the musicians will be playing in the next few beats, in order to coordinate its conducting behavior to their music.

The Virtual Conductor implements a conductor specific 'Behavior Planner' (in SAIBA terms). It manages a flexible conducting animation plan in which (predecessors of) TimedPlanUnits are linked to (predecessors of) TimePegs. When the tempo changes, the values in these TimePegs are updated by the Behavior Planner. The conductor's Behavior Planner adds to, rearranges and removes behaviors from the plan continuously.

The Virtual Conductor's Behavior Planner functionality inspired several specification and architecture components in Elckerlyc. The concept of alignment to predictions inspired the Anticipator. Elckerlyc manages a flexible behavior plan for the SAIBA Behavior Planner and provides it with specification mechanisms (through BML) to specify the removal and addition of behavior.

8.1.3 The Interactive Virtual Dancer

Figure 8.3: Interacting with the Virtual Dancer

The Interactive Virtual Dancer (see Figure 8.3) is a virtual dancer that invites a real partner to dance with her [230]. The Virtual Dancer dances together with a human 'user', aligning its motion to the beat in the music input. The system observes the movements of the human partner by using a dance pad to register feet activity and the computer vision system to gain information about arm and body movements. By responding to the way the human user is dancing, the virtual dancer implicitly invites the user to react to her as well. At any point in time, the virtual human in this application is both expressing herself (dancing to the beat), and perceiving the user's style of dancing. When the virtual dancer is in a 'following' mode she will break off dancing moves when they no longer fit in with the users style and continue with better fitting moves. When in 'leading' mode she may introduce dance moves with a completely new style in order to evoke reactions from the user.

To align its dance moves to the beat of the music, the Virtual Dancer requires *prediction* of the tempo of the music it dances to. To this end, it uses the tempo predictor originally developed for the Virtual Conductor. Dancing is animated using

motion captured dance moves. These dance animations are annotated with sync points that indicate where they should align with the beat in the music, and with movement features (e.g. hands high, hands wide, feet wide, etc.). The latter annotations are used in dance movement selection. Like the Virtual Conductor, the Virtual Dancer requires a flexible behavior plan, in which behavior is continuously retimed (to the beat of the music) and replaced by other behavior (to allow transition to another dance move). The Virtual Dancer features a specialized Behavior Planner (in SAIBA terms) that manages such a plan. Elckerlyc offers SAIBA Behavior Planners of continuous interaction applications with standardized mechanisms to manage such a flexible plan, so that similar plan management and Anticipator functionality can be shared by each of them.

Transitions between dance animations were originally made using a combination of interpolation and a procedural stepping motion. These transitions contained less natural motion than the recorded dance movements, so the fewer transitions occur, the more natural the motion would look. However, early experiments showed that for a user to perceive that the Virtual Dancer acted in reaction to her movement, rapid movement changes of the Virtual Dancer were essential. We needed to set up a trade-off between the naturalness of long motion capture movements with few transitions and the control (specifically: responsiveness) needed for rapid movement changes. To this end we used long ($\approx$ 20 second) mocap animations that were annotated with transition points, indicating natural transition moments within the animation. If no change in dance movement was needed, the dancer could simply execute its long mocap animation. However, if user behavior required a rapid change in the dance movement of the dancer, she could rapidly do so by transitioning to another dance move at the first possible transition point in her current dance animation.

8.2 Design Concerns

8.2.1 Continuous Interaction

Elckerlyc implements novel techniques to support *real-time continuous interaction*. This requires the Realizer to be capable of adaptation – in content and in timing – to the dynamics of the environment and the user. Time adaptations often require synchronizations to *predicted* time events (for example the predicted end of a user's speaking turn, or predicted key time moments in a user's exercise motion, see Chapter 6.5 for some detailed scenarios). This Chapter discusses the Anticipator; a generic architecture element that provides such predictions. Elckerlyc's flexible behavior plan representation that allows flexible microadjustments in the timing of behavior is discussed in Chapter 7.

8.2.2 Naturalness and Control

In Chapter 2 several control aspects were identified. For a virtual human that communicates with a user using gesture and speech, *expressiveness* and *precision* are important: gesture typically requires many control parameters, and is tightly synchronized with speech. Continuous interaction adds *responsiveness*, that is rapid reaction to the behavior of interlocutors, as another required control aspect. The animation paradigms used should provide control parameters that allow *intuitive* control. For different motion types, different control parameters are intuitive. For example, for a gesture, parameters that modify the trajectory (e.g. amplitude, spatial extend) can provide intuitive control, but for a motion describing a loosely hanging arm parameters such as muscle stiffness and damping are more intuitive.

The procedural animation used in gesturing virtual humans often lacks naturalness. The generated motion does not involve the whole body in a coherent manner[2] and does not seem to respect the laws of physics. Elckerlyc's animation system is designed to allow the mix of the physical naturalness provided by physically realistic animation with the control provided by procedural animation. This allows one to mix procedural arm and head gestures with physical simulation of the rest of the body. The forces generated by the gesturing body parts are transfered to the physically simulated body parts, thus creating whole body animation that appears to respect the laws of physics in a more believable manner and that is internally coherent (that is: the movement of the physically steered body parts is affected by the movement of the procedurally steered ones).

8.2.3 Abstraction, Modularity and Extensibility

Virtual human platforms such as Elckerlyc are used by a multi-disciplinary research community – consisting of computer scientists, (computational) linguists, psychologists, and others – interested in studying multimodal human behavior in human-human and human-machine interaction. Using quick prototyping of gestures, facial expressions and body movements, they build virtual humans that display human-like behavior. This multi-disciplinary community has a very wide range of requirements with respect to the *level of abstraction* provided by, for example, Elckerlyc. Some users need access to its functionality only in terms of the *possible behaviors* that the virtual human can display, abstracting away from the details of underlying animations. Others need to exercise detailed control over the *exact form* of the behaviors that they want to study. To achieve such a separation of concerns, HMI has at a very early stage joined the SAIBA (Situation, Agent, Intention, Behavior, Animation) initiative, and in particular the development of the emerging Behavior Markup Language standard (BML) [152, 298]. The SAIBA initiative provides, amongst other things, a view on the architectural issues of building a fully functional virtual human with different layers of abstraction. Within this context, BML is a markup language that allows one to specify the different behaviors that a virtual

[2]Typically some Perlin noise is used to move body parts that are not involved in the gesture in a random manner unrelated to the movement of the gesturing body parts.

human should execute (such as speech, gestures, poses, and gaze), together with their synchronization.

An application that uses a virtual human as one of its components might have several requirements for the BML Realizer. Specific additional gestures and face expressions might be needed; the application might need to run distributed over several machines; the experimenter might need detailed logs of everything that the virtual human does; one might want to replace the graphical embodiment of the virtual human, or its voice; the embodiment of the virtual human might need to reside in a custom game engine instead of in Elckerlyc's default render engine; and one might need to plug in completely new custom behaviors and modalities for a specific usage context.

Developing extensions or alternative configurations of Elckerlyc should be possible without requiring changes to the core Elckerlyc system (that is, extensions should not require re-compilation of the Elckerlyc source). After all, if Elckerlyc extensions lead to a modification of Elckerlyc itself, then this would essentially lead to a separate Elckerlyc fork for every application using Elckerlyc. This would make it difficult to share new extensions with the community. Also, once Elckerlyc has been forked to accommodate a new modality engine or behavior type, it becomes difficult to take advantage of improvements in the 'core' Elckerlyc source: they need to be painstakingly merged into the fork.

Below follows a number of requirements for Elckerlyc, concerning possibilities for extension or adaptation, that will be discussed later in more detail. These are all what we call *non-invasive modifications*: Some of them entail the implementation of new *run-time libraries*, others only the addition of new *resources*; but none of them require *compile time* dependencies for Elckerlyc on new code.

- On the output side, provide integration with new or existing render environments, speech synthesis software, physical simulators, sound generators, etcetera.

- On the input side, provide flexible ways to send BML to the Realizer, and to adapt the BML stream with capabilities for filtering and logging.

- Provide a transparent mapping from input (BML behavior elements) to output (control of the virtual human's embodiment).

- Provide possibilities to add new behavior types or output modalities.

- Provide easy ways to integrate a BML Realizer as a component in an application, independent of variables such as the OS and programming language on which the application is developed.

8.3 Related Work

8.3.1 BML Realization

As already discussed in detail in Chapter 6.2, the Behavior Markup Language (BML) provides a general, Realizer-independent description of multimodal behavior that can be used to control a virtual human. BML expressions (see Figure 8.4 for a short example) describe the occurrence of certain types of behavior (facial expressions, gestures, speech, and other types) as well as the relative timing of the involved actions [152] (see Figure 8.5 for a the standard synchronization points that are used for this).

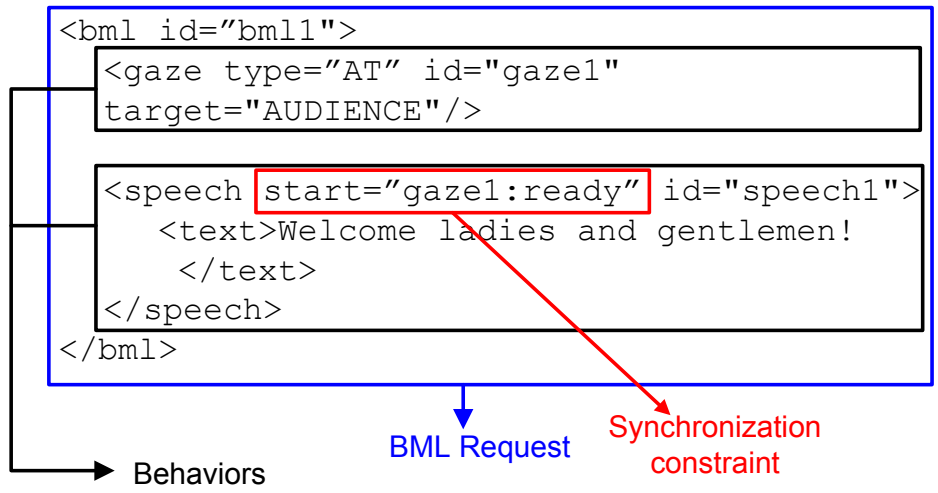


Figure 8.4: An example of a BML request containing a gaze and a speech behavior. A synchronization constraint ensures that the speech starts after the gaze is aimed at the audience.

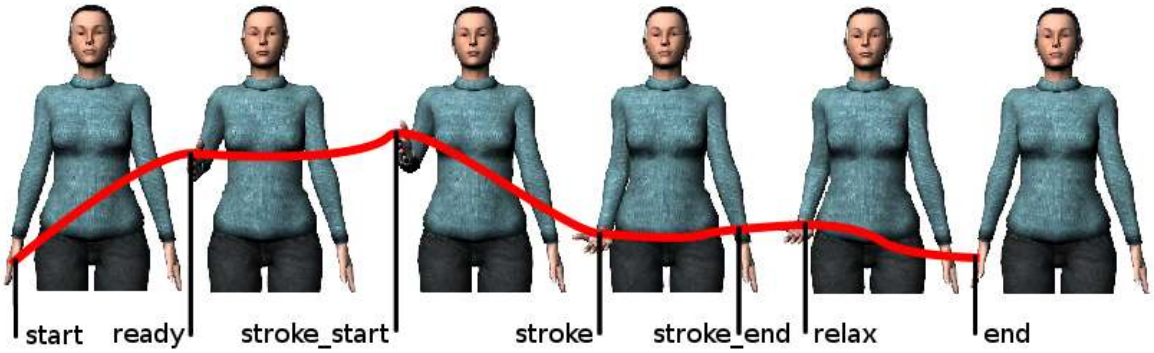


Figure 8.5: Standard BML synchronization points

Elckerlyc is a new BML Realizer. Recently several other BML Realizers have been developed [54, 63, 111, 280, 281], and existing frameworks for multimodal behavior generation are being modified to support BML [104, 155]. These Realizers are typically used in turn-based interaction applications or are (also) employed offline, for instance in the reproduction of annotated behavior of real humans [111].

Since these Realizers are not specifically designed for continuous interaction they do not support timing or shape adjustments of behaviors that are already in the play queue.

Body animation is specified by parameterizable keyframes and end effector movement trajectories [54, 104, 111], procedural controllers that support emotional parameterization [104, 111], keyframe animation created by artists [280], motion capture [54, 111, 280], or using biomechanical models of human movement [54, 155, 280].

Facial animation is typically specified using either MPEG-4 [54, 63, 104] or face morphing [111]. SmartBody [280] is a noticable exception: it allows both face morphing for the animation of visemes and skeleton-based facial animation.

Elckerlyc adds physical simulation as another body animation paradigm. Facial animation in Elckerlyc can be steered by both morph targets and MPEG-4. Elckerlyc allows all these facial and body animation paradigms to be used together, both in parallel and sequentially.

My focus is thus not solely on creating animation using one of these paradigms, but also on designing a Realizer that can make use of – and combine – animation generated by each of them. Such a Realizer provides an adjustable trade-off between the naturalness and control provided by different animation paradigms.

8.3.2 Continuous Interaction

Continuous interaction needs flexible planning and behavior synchronization and anticipation, not only to internal modalities but also to the environment and participants in the interaction.

Thórisson [284] describes an interactive cartoon character engaging in an information exchange with the user. In their Ymir system, perception and behavior generation are parallel and ongoing all the time, and behavior plans can be modified last-moment in response to new perceptions and decisions. In Ymir, this flexibility is achieved by incrementally sending small blocks of behavior to an Action Scheduler that executes them ballistically. Elckerlyc's behavior execution process never enters a ballistic stage and it can therefore achieve a finer interruption and coordination granularity than Ymir.

Loyal et al. [177] describe a system that allows the authoring of highly interactive motion and demonstrate their approach in an interactive game with a personality-rich character. The behavior of this character is tightly coupled to changes in the environment (a bouncing ball, moving mouse, etc.), also with respect to the exact timing. Continuous changes and the unpredictability of this environment require flexible animation planning and execution processes. This is achieved by animating their character using a flexible mechanisms that can interrupt keyframe animation segments and fluently concatenates them.

Like Loyal et al.'s system Elckerlyc offers synchronization to the *anticipation* of user behavior (e.g. what is the tempo that the musicians are playing in). This is not just relevant for games or for a virtual conductor, but also necessary for general conversational capabilities of a virtual human (see Chapter 5, [284]). Elckerlyc

implements more complex behavior realization than [177], adding a modality for speech and physically specified animation and allowing internal synchronization between modalities.

8.3.3 Mixed Dynamics

Mixed dynamics combines the precision of kinematic animation (including procedural animation) on certain selected body parts, with the naturalness of physical simulation on the remaining body parts, in a physically coherent manner. It was pioneered by Isaacs et al. [123]. Mixed dynamics uses inverse dynamics to determine joint torques on kinematically steered body parts. These torques are transferred to the physically steered body parts. In addition to being affected by joint torques from kinematic joints that are connected to physical body parts, the physical body is affected by gravity, collision impulses, friction forces and joint torques from the physical controllers that steer it. Isaacs et al. use a custom designed physics simulator to achieve this. My mixed dynamics algorithm builds on their ideas, and extends them by using efficient iterative techniques to calculate in real time the torques exerted by the kinematically steered joints and by providing easy integration with existing real-time physics simulators. However, unlike Isaacs et al.'s system, Elckerlyc currently requires the physical simulation (if active) to act on only one subtree of the skeleton, which must include the root joint. I refer the interested reader to Chapter 2 for an extensive overview of real-time animation techniques for mixed dynamics and physical simulation and Chapter 3 for a thorough comparison of those methods with mine.

Other hybrid physical simulation/kinematic systems have been designed to allow switches between full-body kinematic animation and full body physical simulation and vice versa, depending on the current situations' needs (see Chapter 2.3.2.4 for an overview). Rather than doing full body switches, I contribute a hybrid method that allows switching to a different *mix* of physically and kinematically steered joints in real time.

8.3.4 Extensibility of Existing Realizers

Like Elckerlyc, the BML Realizers SmartBody [280], EMBR [111] and Greta [104] were specifically designed for integration with an existing renderer, to allow a wide range of behavior types, and/or to facilitate integration in different applications. Elckerlyc additionally contributes a transparent and adjustable mapping from BML to PlanUnits (rather than the mostly hard coded mappings employed in other Realizers), and allows for easy integration of new modalities and embodiments, for example to control robotic embodiments. In this section, I discuss how various requirements were solved for the three Realizers mentioned above, and shortly indicate the differences with Elckerlyc's solutions. Section 8.16 discusses the solutions used in Elckerlyc in detail. Chapter 10 shows some examples of how Elckerlyc's extensibility features impact actual use in new applications.

8.3.4.1 Integration with Existing Renderers

SmartBody provides the BoneBus library to connect the SmartBody Realizer to a renderer. BoneBus is a C++ library that uses UDP to transport (facial) bone positions and rotations from the Realizer to the renderer. BoneBus is designed to hide the details of the exact communication protocol used, so that its exact implementation can be changed at a later stage without changing Realizers or renderers that use the library. Because the data transport protocol is non-trivial and due to change, re-implementing BoneBus in programming languages other than C++ or using the BoneBus interface with other transport mechanisms (TCP/IP, shared memory, etc.) is infeasible. Currently, SmartBody has been integrated with the Unreal 2.5[3] and Panda3D[4] (in CADIA's branch of SmartBody) renderers; partial integrations are available for Ogre[5], Gamebryo[6] and Half-Life 2.[7]

The output of Greta contains MPEG-4 facial and body action parameters. By using the MPEG-4 standard, Greta can potentially be used with any renderer that supports MPEG-4. However, MPEG-4 –especially for body animation– is not widely supported by existing renderers.

Elckerlyc currently uses the Thrift remote procedure call (RPC) framework [272] to handle its communication with the renderer. Unlike the BoneBus library, this allows the setup of a communication channel that is agnostic to the programming language used on either side and that allows one to configure and change the mode of transport (e.g. TCP/IP, shared memory, pipes).

8.3.4.2 Available Behavior Types and Extensibility

SmartBody uses both keyframe animation and a fixed set of biologically motivated motion controllers (e.g. for gaze) to achieve facial and body motion. EMBR uses keyframe animation, procedural animation with a fixed set of expressive parameters (comparable to those used in Greta), autonomous motion (such as eye blink and balancing), morph targets for facial animation, and controllable shaders (e.g. for blushing). Greta uses procedural body animation that is parameterized by a fixed set of expressivity parameters. Facial animation is specified using Ekman's action units [71].

Elckerlyc allows the use of all of the above animation types, and adds physically simulated animation and audio (sound effect) behaviors. More importantly, we contribute the ability to add custom behavior types and new output modalities *without* requiring modifications to Elckerlyc's source code, described in Sections 8.16.1 and 8.16.2.

[3] `http://www.unreal.com/`
[4] `http://www.panda3d.org/`
[5] `http://www.ogre3d.org/`
[6] `http://www.emergent.net/`
[7] `http://www.valvesoftware.com/`

8.3.4.3 Integrating the Realizer as a Component in an Application

SmartBody offers integration with the Active MQ[8] messaging system to provide independency of platforms and programming language, and to allow running the Realizer on a separate machine. EMBR and Greta offer integration with the SEMAINE/Active MQ [257] messaging frameworks to achieve this; Greta additionally offers integration with Psyclone.[9]

Elckerlyc uses Ports and Adapters to facilitate quick development of support for new types of integration; current implementations include support for the SEMAINE/Active MQ system and a simple direct TCP/IP connection. Section 8.16.3 discusses this in detail, and touches upon several other things made possible by this architectural feature.

8.4 Example Application

HMI's Virtual Conductor (see Section 8.1.2) will be referred to throughout this chapter as a running example explaining aspects of the Elckerlyc BML Realizer. Although it was originally built when Elckerlyc had not yet been developed, the core elements of the virtual conductor have found their way into Elckerlyc. Video material showing elements described in the running example can be found at `http://thesis.herwinvanwelbergen.nl/`.

Running example 1: The Virtual Conductor

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

Of course, any virtual human needs to be able to plan multi-modal behavior, and to extend or change the planned behavior based on its perceptions. In conversations, people also subtly adapt their timing to each other (see Chapter 5). Currently, HMI is developing a research application that will be very close to the virtual conductor with respect to these timing changes: the reactive virtual trainer. The virtual (fitness) trainer will perform an exercise together with a human user in a certain tempo. Since the user will not necessarily perform the exercise with robotic precision, the virtual trainer needs to be able to adapt the timing of its behavior (the

[8] `http://activemq.apache.org/`
[9] `http://www.cmlabs.com/psyclone/`

movements of the exercise as well as accompanying explanations and motivational utterances) to the timing with which the user performs the exercise.

8.5 Architecture

Elckerlyc's architecture is based upon the SAIBA Framework [152], which contains a three-stage process: *communicative intent planning*, multimodal *behavior planning*, resulting in a BML stream, and *behavior realization* of this stream (see also Chapter 5.4). Elckerlyc encompasses the realization stage.

Running example 2: The Virtual Conductor

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

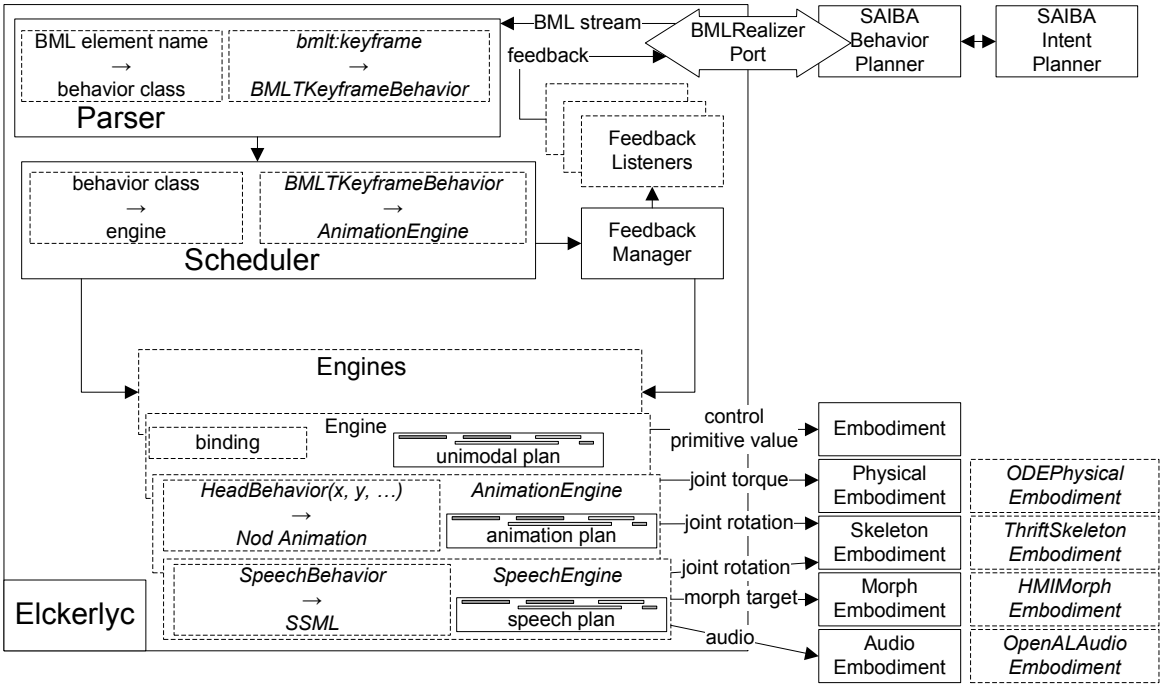


Figure 8.6: Architecture of the Elckerlyc BML Realizer, together with the SAIBA Intent Planner and the SAIBA Behavior Planner which make up the other two parts of the SAIBA framework

Figure 8.6 shows Elckerlyc's global architecture, and indicates its relation to the overall SAIBA framework. The main information flow goes from the SAIBA Behav-

ior Planning module to the Realizer, in the form of a BML stream. A feedback loop communicates backwards from the Realizer to the SAIBA Behavior Planner. The Realizer notifies the SAIBA Behavior Planner when a behavior is successfully executed and warns it if a particular behavior cannot (or: no longer) be executed. The latter typically occurs when unexpected events occur that prohibit successful realization or, in Elckerlyc, when predictions of user behavior are revised drastically (see Chapter 7.3.6). The SAIBA Behavior Planner will have to deal with situations where the original plan is no longer feasible.

Behavior scheduling determines the synchronization of the behaviors within and between modalities and distributes the different behaviors over appropriate Engines (e.g. SpeechEngine, AnimationEngine). Each Engine subsequently executes the behaviors through one or more *Embodiments* (for example, a SkeletonEmbodiment containing the skeleton representation of the virtual human or an AudioEmbodiment containing the audio channel it may use), using the appropriate control primitives values for that embodiment (for example: joint rotations for SkeletonEmbodiment, audio for the AudioEmbodiment).

The Engines act independent of each-other; the timing constraints between the behaviors they manage is maintained through the PegBoard. One exception to this is the TTSEngine, which requires access to the AnimationEngine and the FaceEngine to add the speech animations (lip and mouth movement) to their (facial) animation plans. Some Engines (e.g. the InterruptEngine) have access to the Scheduler, allowing them to change or remove ongoing behavior in other Engines.

8.5.1 Continuous Interaction

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

Running example 3: The Virtual Conductor

In the continuous reactive and anticipatory interaction between conductor and ensemble, the behavior plan that was initially constructed from the score needs to be adapted all the time. Some changes merely require re-timing, e.g. in order to provide tempo feedback to the ensemble, or can be solved simply by adding an extra behavior: add an appreciative nod or smile; conduct two-handed when the ensemble is not paying attention. Other changes are larger and require substantial modifications. An example of the latter is when the conductor stops in the middle of the piece, because the ensemble completely messed up the music. In this chapter I focus on the timing adaptation of ongoing behavior, since that requires specific capabilities in the Realizer.

As Running Example 3 suggests, some changes to scheduled behavior only concern the timing, and should not lead to completely rebuilding the animation plan. Small adaptations of the timing of scheduled behavior are not specific to conduc-

tors, but also occur in other interactions. Elckerlyc offers detailed temporal control over the execution of scheduled behavior.

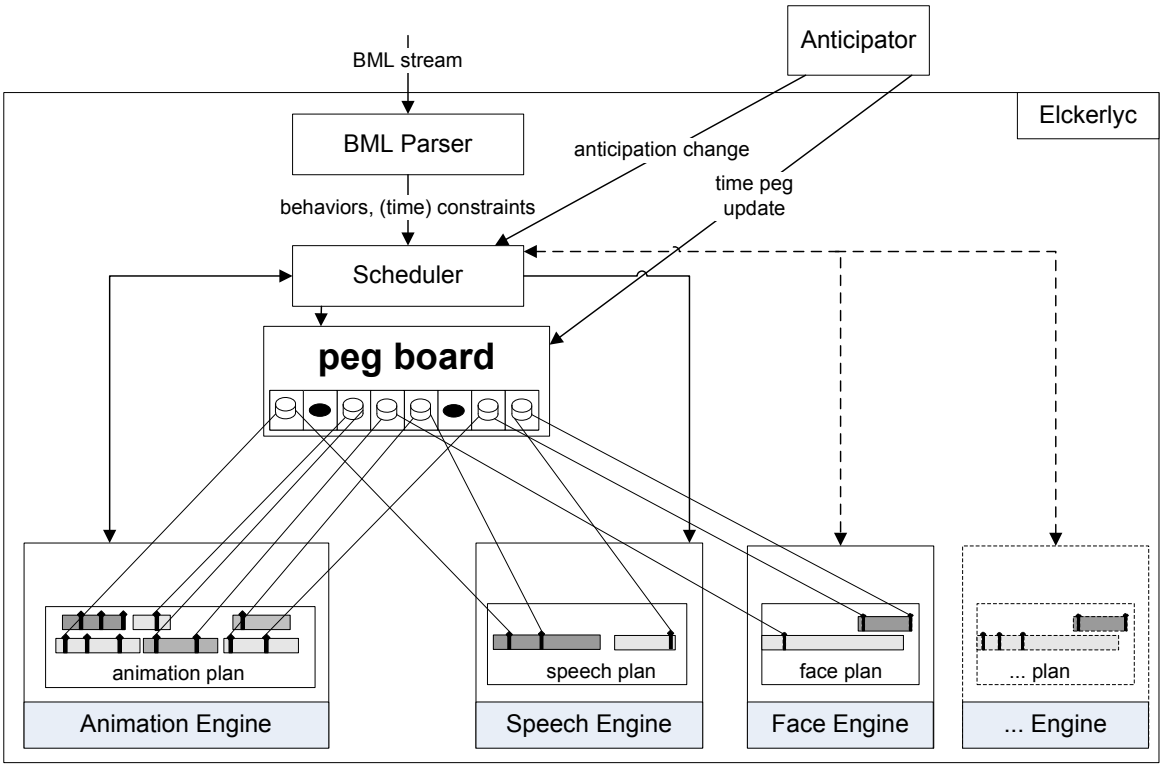


Figure 8.7: TimePegs, Anticipators and Multimodal Synchronization in Elckerlyc.

To achieve this detailed temporal control, I introduced *TimePegs* and *Anticipators* (Figure 8.7). The behavior of the virtual human is specified in a stream of BML elements (called *behaviors*). Synchronization of the behaviors to each other is done through BML *constraints* that link synchronization points in one behavior (start, end, stroke, etc; see also Figure 8.5) to synchronization points in another behavior. The *Peg Board* maintains a list of TimePegs – symbolically linked to those synchronization points that are constrained to be on the same time – together with the current expectation of their actual execution time (which may change at a later time). Interaction with the world – and conversation partner – is achieved through Anticipators. An Anticipator instantiates synchronization points that can be used in the BML stream to constrain the timing of behaviors. It uses perceptions of events in the real world to update the corresponding TimePegs, by extrapolating the perceptions into predictions of the timing of future events.

Running example 4: The Virtual Conductor

For the virtual conductor, an Anticipator has been implemented that predicts the tempo with which the musicians play the music (using audio processing algorithms described elsewhere [229]). It provides TimePegs to Elckerlyc that represent the predicted beats. The BML stream for the conductor specifies the appropriate conducting gestures for a

piece of music, and their timing. If the musicians play too slowly, the conductor would like to increase the tempo. A subtle technique to achieve this is to conduct in the same tempo as the musicians are playing, but slightly ahead of them, so they constantly have the feeling of being 'too late'. This is done by aligning the conducting gestures to the TimePegs for the predictions of the Anticipator, but slightly ahead of them (see also BML Example 31).

BML Example 31 Example script which uses a conducting Anticipator. Two synchronization points in the conduct1 behavior (conduct1:start and conduct1:beat2) are synchronized to desired conducting beats provided by the Conducting Anticipator.

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

For smooth turn-taking purposes, an Anticipator could be designed that can predict relevant positions to take the turn from user behavior observed through sensors. However, designing these kinds of Anticipators is challenging, no off-the-shelf systems are available.[10] To demonstrate and test Elckerlyc's continuous interaction capabilities, I have implemented a few simple Anticipators. The Metronome Anticipator, as demonstrated in the webstart, 'predicts' beats of a metronome. The metronome tempo can be adjusted in the user interface while the behavior (linked to its TimePegs) is playing, modifying the timing of this linked behavior in real time. A Spacebar Anticipator was created for specifying behavior that reacts immediately to a spacebar press or release. Unlike the previously mentioned Anticipators, the spacebar Anticipator does not provide event prediction, but simply sets the time of a TimePeg to the time of a spacebar press or release.

Other, more application specific, Anticipators were developed for the Virtual Trainer (to predict the user's exercise tempo) and for the Virtual Conductor (to predict the musician's playing tempo). These are discussed in Chapter 10.

[10] See Chapter 12.2.2 for possible implementations.

8.5.2 Naturalness and Control

8.5.2.1 Body Animation

Elckerlyc makes use of motion captured motion, procedural animation, and physical simulation to steer the movement of the virtual human. Physical simulation provides physically realistic motion and (physical) interaction with the environment. In Chapter 2, I argue that physical realism is one of the aspects of natural motion. Physical controllers can robustly retain or achieve desired movement states (joint rotation, center of mass position, etc.) under the influence of external perturbation. This robustness comes with a disadvantage: precise timing and limb positioning using physical controllers is an open problem (see also Chapter 2.3.1.5). Procedural animation offers precise timing and limb positioning and can make use of many motion parameters. However, it is hard to incorporate movement details such as those found in recorded motion into the mathematical formulas that steer procedural animation. Furthermore, to maintain physical realism, it has to be explicitly authored in the procedural model for all possible parameter instances. Motion (capture) editing techniques retain the naturalness and detail of recorded motion. However, these techniques produce natural motion only when the modifications are small, and the number of required recordings grows exponentially with the number of motion parameters used. Chapter 2 discusses the naturalness and control offered by different animation techniques in greater detail.

Elckerlyc offers a mix between, on the one hand, the *precise temporal and spatial control* offered by kinematic animation techniques such as motion capture, keyframe animation and procedural animation, and, on the other hand, the physical realism of physical simulation. Elckerlyc can steer a virtual human using kinematic animation and physical simulation. These two paradigms can be used in parallel on different body parts, and the assignment of body parts to one of the paradigms can be modified on the fly. I model the force transference from body parts that are kinematically animated to the physically simulated part of the body, increasing the perceived physical realism and physical coherence of the resulting motion. The possibility to specify animation both kinematically and physically also allows one to select the paradigm that is the most *intuitive* to author for each required animation. Physical interaction with the environment, for instance balancing, is hard to author procedurally, but relatively easy to author using physical controllers in a physically simulated environment. Gestures, on the other hand, need to adhere to strict timing and spatial constraints and typically make use of many control parameters. They are therefore better defined procedurally (see also Chapter 2).

8.5.2.2 Facial Animation

Elckerlyc can make use of morphing and MPEG-4 facial animation simultaneously. Morphing involves interpolation between the vertex positions of different face meshes of the same face designed by an artist. For each new face expression a new mesh needs to be created. Furthermore, morph-based animation cannot be shared between different faces. The MPEG-4 facial animation standard describes animation

as the movement of facial feature points. The feature point locations and movements are defined in a face independent manner, so that MPEG-4 configurations can be reused on different faces. This allows the use of automatic mappings from face muscle movements and emotions to MPEG-4 (see also Section 8.8). However, animation created using such an automatic mapping is often less natural than morph-based animation, because the morphs are designed by skilled artists that were able to take face specific idiosyncrasies into account.

In typical application scenarios of Elckerlyc, morphing is used whenever a small set of facial configurations suffices; for example for eye blinking and viseme (visual phoneme) animation. Other face animations are implemented using MPEG-4.

8.5.3 Modularity and Extensibility

Modularity and extensibility were explicit requirements in the design of Elckerlyc. Figure 8.6 shows the relevant parts of the architecture for extensibility and modularity. Dashed boxes indicate components that can be changed at initialization, black boxes indicate unchangeable components.

The SAIBA Behavior Planner controls the virtual human by sending a stream of BML Blocks to Elckerlyc through a BML Realizer Port. Section 8.16.3 discusses how Ports can be used, for example, to integrate Elckerlyc with various distributed messaging systems. The *Parser* parses the incoming BML stream. It provides the Scheduler with a list of BML behavior elements and time constraints between these elements. Section 8.16.1 discusses how to add custom BML behavior elements. The Scheduler generates an execution plan, based on these elements and constraints. Different Engines (e.g., a TTSEngine, an AnimationEngine, a FaceEngine) keep track of and manage unimodal plans for their specific modality. Section 8.16.2 discusses how to add new Engines. The Scheduler does not need to know about the inner workings of an Engine (other than that it implements the Engine interface defined in Chapter 8.6), it only knows which Engine was registered for certain behaviors. This allows one to add or replace Engines easily. Engines are also responsible for translating behavior elements to a form that is actually displayed on the Embodiment of the Virtual Human. Section 8.6.1 discusses how this mapping from abstract behavior element to concrete forms can be configured in a flexible manner within an Engine. Section 8.16.4 discusses how to connect a new Embodiment to an existing Engine.

8.6 Engines

In the Scheduling stage (see Figure 8.6), multimodal behavior described in a BML stream is converted into Plans that are to be executed by different Engines, as already indicated in Section 8.5.3. The Scheduler has access to several Engines (e.g. SpeechEngine, AnimationEngine, see also Figure 8.6) with which it communicates only through the interface defined below. The scheduler knows for each behavior type which Engine handles it. Each Engine must implement the Engine interface

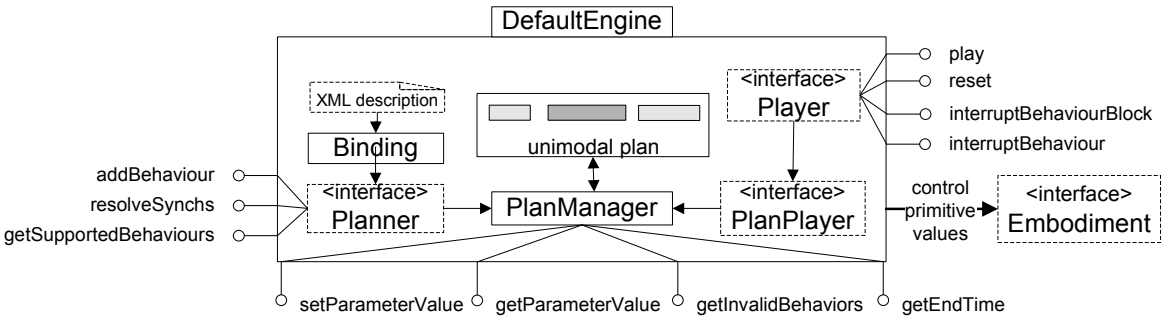


Figure 8.8: The architecture of an Engine. Dashed objects are configured at initialization time.

(indicated by the lollipops in Figure 8.8): an Engine is required to implement functionality to:

1. Provide a list of BML behaviors types the Engine supports.
2. Add a BML behavior to its Plan.
3. Resolve unknown time constraints on a BML behavior, given certain known time constraints.
4. Check which (if any) behaviors in the Plan are currently invalid.
5. Play the current Plan (at a certain time).
6. Interrupt a behavior.
7. Interrupt all behaviors in a BML block.
8. Get the (predicted) end time of a behavior in the behavior plan.
9. Get the current value of a parameter in a certain behavior.
10. Set the value of a parameter in a certain behavior.

Functions 1-4 are used in the Scheduling process, which is discussed in detail in Chapter 7.3.5.1. Functions 6-10 provide the functionality required to continuously modify the multimodal behavior plan as it is being played.

Elckerlyc's current Engines are implemented on the basis of the DefaultEngine, a skeleton implementation of the Engine interface. The DefaultEngine uses a Planner, PlanManager, Player and PlanPlayer and manages and plays a unimodal Plan containing TimedPlanUnits (e.g. a gesture, a speech clause, etc.). The Planner resolves and constructs the unimodal Plan on the basis of provided behavior elements and the constraints acting upon them. The PlanManager manages the unimodal Plan and provides several functions to query its state or modify it. The Player plays back the units in the Plan. In the DefaultPlayer, this functionality is fully delegated to a PlanPlayer. The AnimationEngine requires a specialized Player that manages the combination of TimedPlanUnits that act simultaneously on the virtual human (e.g. physical simulation and keyframe animation), but it can still delegate most of

its execution functionality to a PlanPlayer. The DefaultPlanPlayer provides a simple default implementation of a PlanPlayer. For TimedPlanUnits whose execution would otherwise block the calling thread, a NonBlockingPlanPlayer was designed to play the TimedPlanUnits in a separate thread.

8.6.1 Bindings

BML provides abstract behavior elements to steer the behavior of a virtual human. A specific BML Realizer is free to make its own choices concerning how these abstract behaviors will be displayed on the virtual human's Embodiments. For example, in Elckerlyc, an abstract 'beat gesture' is by default mapped to a procedural animation from the Greta's gesture repertoire. The developer may instead want to map this beat gesture behavior to a different form, for example, to a high quality motion captured gesture. The mapping of BML behavior to TimedPlanUnits in the Plan is typically configured in an XML representation: the Binding. This allows one to change the mapping from BML behavior to TimedPlanUnits without programming, by simply modifying an entry in the Binding.

8.6.2 Control Primitives and Embodiments

Engine	Control Primitive	Embodiments
AnimationEngine	joint rotation/translation	SkeletonEmbodiment
	joint torques, root force	PhysicalEmbodiment
FaceEngine	morph targets	MorphEmbodiment
	MPEG-4 FAPs	MPEG-4 Embodiment
TTSEngine	audio	AudioEmbodiment
	joint rotations	SkeletonEmbodiment
	morph targets	MorphEmbodiment
	MPEG-4 FAPs	MPEG-4 Embodiment
TextEngine	text	TextEmbodiment
AudioEngine	audio	AudioEmbodiment
WaitEngine	-	-
InterruptEngine	scheduler method	-
PVCEngine	scheduler method	-

Table 8.1: Elckerlyc's Engines, and the Control Primitives they use to steer their Embodiments.

Execution results in values that are set for Control Primitives in an Embodiment that is steered by the Engine. For example, the execution of animation results in joint rotations on a SkeletonEmbodiment. The Control Primitives and Embodiment interface that is used within one Engine is fixed. However, the *implementation* of the Embodiment interface can be selected at initialization time. For example, the FaceEngine uses an MPEG-4 Embodiment interface and can be configured to use the 'standard' implementation of this interface provided by Elckerlyc, or to use

an MPEG-4 Embodiment implementation that sends the FAP values to the XFace [15] talking head. Table 8.1 provides an overview of the Engines implemented in Elckerlyc and the Control Primitives and Embodiments used by them.

8.7 The AnimationEngine

One of the Engines introduced in Section 8.5 is the AnimationEngine, consisting of an AnimationPlanner, that creates an AnimationPlan, and an AnimationPlayer (Figure 8.9) responsible for executing it. The plan contains descriptions of both physical and procedural motions. The AnimationPlayer implements and combines several existing state-of-the-art techniques for procedural animation and physical simulation. The mixed dynamics system described in Chapter 3 is used to execute the mix of physically and kinematically specified motions simultaneously in a physically coherent manner. In addition, I have implemented smooth and automatic switching between physical and kinematic steering (or vice-versa) on parts of the virtual human's body whenever this particular mix changes. The end result is an integrated animation of the virtual human which still allows one to adapt the precise timing by adjusting the TimePegs described in Section 8.5.1.

8.7.1 Mixed Dynamics

In many situations, different positive features of procedural motion and physical simulation are needed simultaneously, but acting on different body parts. To achieve this, the AnimationPlayer dynamically assigns a body structure to the virtual human that is divided in a *physically* steered part and zero or more *kinematically* steered parts.

Running example 5: The Virtual Conductor

The conducting gesture of the right hand clearly needs the tight temporal control that is best achieved with procedural motion. On the other hand, the balanced pose, and the left arm hanging loosely down in gravity, are best left to the realism of physical simulation. The two will necessarily affect each other: the often quite vigorous movements of a conducting motion should have a perceivable impact on the dynamics of the rest of the virtual human (see also the movies on the web page of this thesis).

Each part is a tree-structure of joints, connected by rigid bodies. In the current implementation, the physical body part must contain the root joint (or be completely empty). See Figure 8.10 for an example set of such body structures.

Running example 6: The Virtual Conductor

For the current gesture repertoire of the virtual conductor, the three body configurations shown in Figure 8.10 are needed: the left-most is used when the virtual human stands in a balanced pose and makes gestures simultaneously with both hands; the middle one is used for making gestures with the right hand only (letting the left hand hang loosely

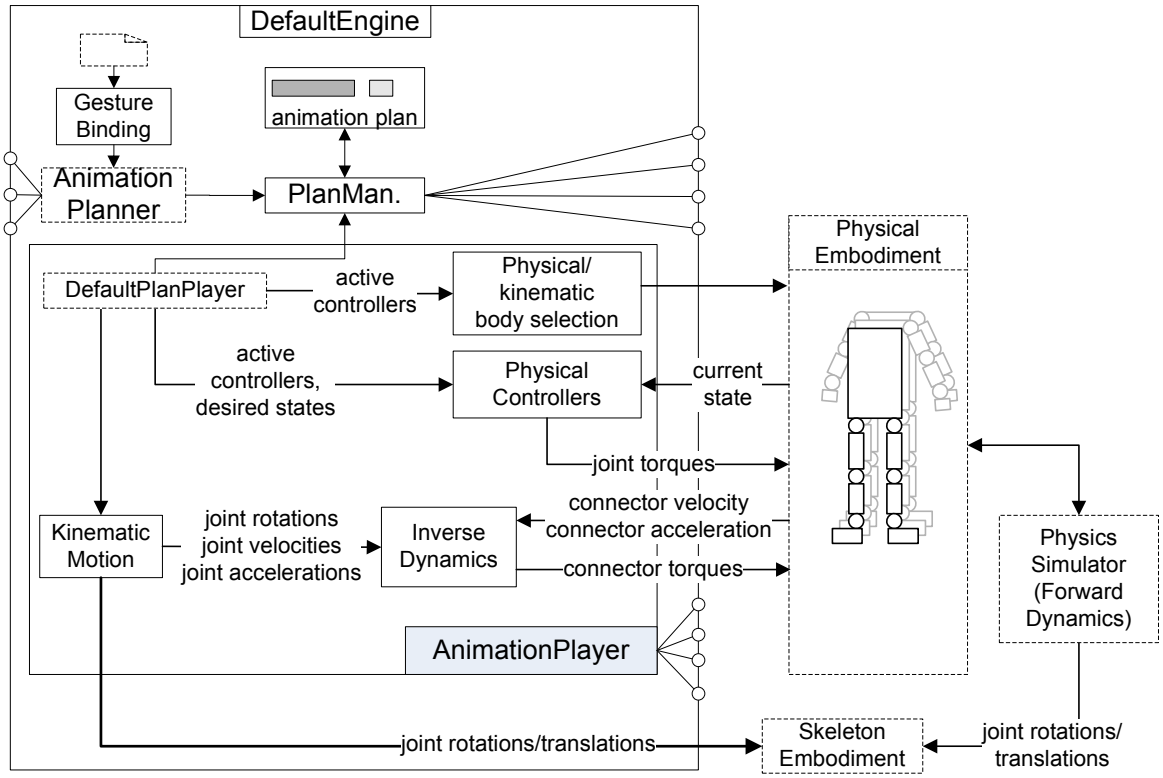


Figure 8.9: The AnimationEngine

down in simulated gravity using a physical controller); the right-most one is needed when the conductor is not gesturing at all.

In the AnimationPlayer (see Figure 8.9), motion is executed by MotionUnits (see Chapter 4.2) which are specified either kinematically or physically. The motion can be adapted in real time by changing the parameters and timing of a kinematic MotionUnit or the desired state of a physical MotionUnit, respectively.

Kinematic motion directly rotates selected joints in the SkeletonEmbodiment. The joint rotations, angular velocities and accelerations of the kinematically steered body parts result in a torque. This torque is calculated using inverse dynamics and applied to the physical body. The physical controllers in turn apply joint torques that aim at reducing the discrepancy between the desired physical state of the physically steered body part and its current physical state (see Chapter 4.2.3).

8.7.2 Switching Physical Representation

As can be seen from Running Examples 5 and 6, the specific configuration of kinematically steered and physically steered joints that is active at a certain moment may need to change when the virtual human performs different behavior. A switch from kinematical to physical control on some body part K is implemented by augmenting the set of physically controlled parts P with the rigid body representation

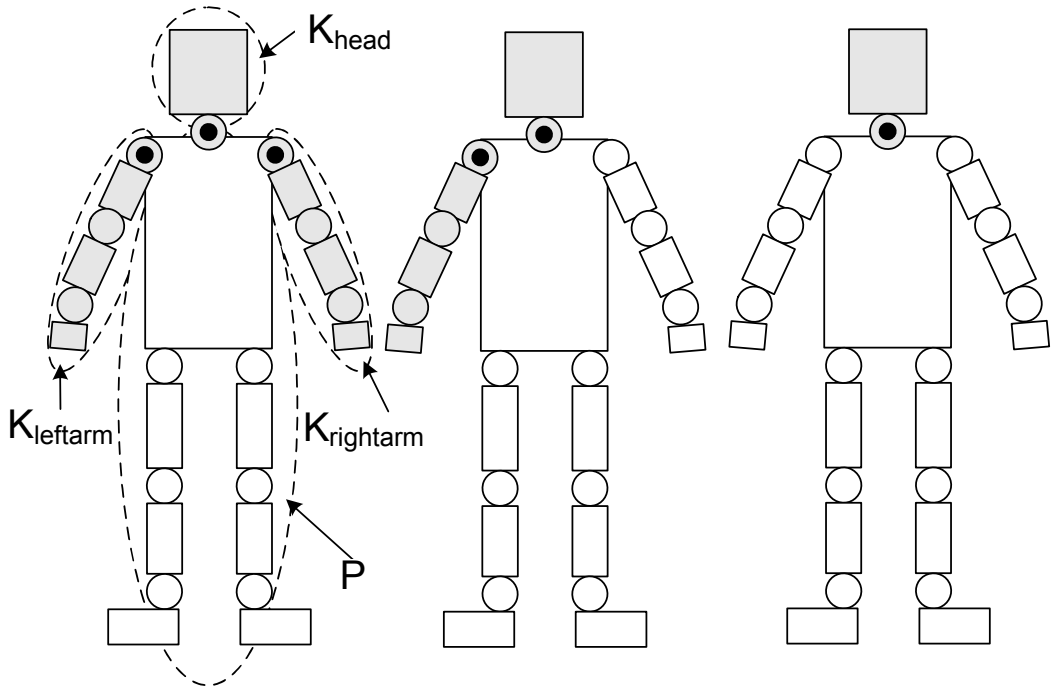


Figure 8.10: Left: a body divided into kinematic parts K that steer the arms and head and a dynamic physical part P that steers the lower body and trunk. Several variations of K and P are used in the conductor, this figure shows three of them.

of K and applying the current joint velocity and rotation to the matching joints in the new physical representation. Before the switch, my mixed dynamics approach applied a torque calculated from the movement and position of the joints in K to P. Augmenting P with an articulated set of rigid bodies with joint velocities and rotations that are the same as the joint velocities and rotations in K will obviously result in a similar torque on P. Therefore such a switch results in a smooth transition.

Finally, a physics simulator[11] is used to calculate the actual resulting rotations of the physically steered joints. I refer the interested reader to Chapter 3 for the implementation details of my mixed dynamics method.

A switch from the physical to kinematic control removes the physical representation of the body part from the physical body of the virtual human and inserts a new kinematic part K. If the movement on K directly after the switch is similar to the movement in its former physical representation, no sudden forces occur on the new physical body. To achieve this, the kinematic MotionUnit steering K must smoothly connect to the former physical motion. A simple way to achieve this is by specifying a transition MotionUnit in BML to connect the physical motion to the kinematic motion. Elckerlyc's Hermite quaternion spline transition MotionUnit generates a C^2 continuous curve between the end configuration of the joint in the physical body and the start configuration of the kinematic motion that is to be executed on the joint [143]. This curve approximates a rotation path with minimal total angular ac-

[11] Elckerlyc currently uses the Open Dynamics Engine: `http://www.ode.org`, but any other real-time physics simulator could be used

celeration from one joint rotation to another, and satisfies the start and end angular velocity constraints prescribed by the physical configuration at the time of the switch and the start configuration of the next kinematic MotionUnit. Alternatively, one can design a transition MotionUnit that ensures smooth *end effector* (hand) movement [155] or make use of *procedural* MotionUnits with a flexible start state [104, 155] to ensure a smooth transition.

8.7.3 Planning Animation

The AnimationPlanner is responsible for selecting MotionUnits based on their BML specification. These MotionUnits are instantiated (that is, their timing and parameter values are assigned) by the AnimationPlanner and then inserted into the AnimationPlan. I call such instantiated MotionUnits *TimedMotionUnits*. This AnimationPlan is played by the AnimationPlayer described in Section 8.7. The Planner makes use of the *GestureBinding* to select MotionUnits. The GestureBinding is an XML specification describing how a BML behavior is bound to a specific MotionUnit, possibly constrained by parameters of the BML behavior, and maps parameters in BML to parameters in the MotionUnit (See Figure 8.11). This allows one to bind a new BML behavior to a MotionUnit without changing Elckerlyc itself and to easily exchange the exact realization of a behavior (by binding it to a different MotionUnit). Currently, a BML behavior specification is bound to at most one MotionUnit.

The AnimationPlanner assists the Scheduler by resolving the execution time of unknown TimePegs for a MotionUnit, given certain known time constraints on that MotionUnit (see Figure 8.12). For each procedural MotionUnit a preferred duration is specified within the definition of the MotionUnit itself. Time constraints and requested motion times are linked to the keys of a MotionUnit. If only *one time constraint* is specified on the MotionUnit, the requested motion times are resolved in such a way that the specified time constraint is satisfied and the duration of the MotionUnit is its preferred duration. If *two or more time constraints* are set on a MotionUnit, uniform scaling is applied to the motion phases between constrained keys. The timing of *start and end keys*, if not constrained, is set to maintain the average stretch or skew (as compared to the preferred duration, see also Figure 8.12) caused by the specified time constraints. Note that if *exactly two* time constraints act upon the MotionUnit, the above strategy is equivalent to the uniform scaling used in SmartBody [280]. The current stretching/skewing strategy does not have a biological basis.

8.7.4 Checking the Validity of the AnimationPlan

Validity checks on behaviors in the AnimationEngine are TimedMotionUnit specific. For example, in a procedural animation, the minimum and maximum durations are specified, and the validity check fails if the behavior is stretched or skewed beyond these specified limits.

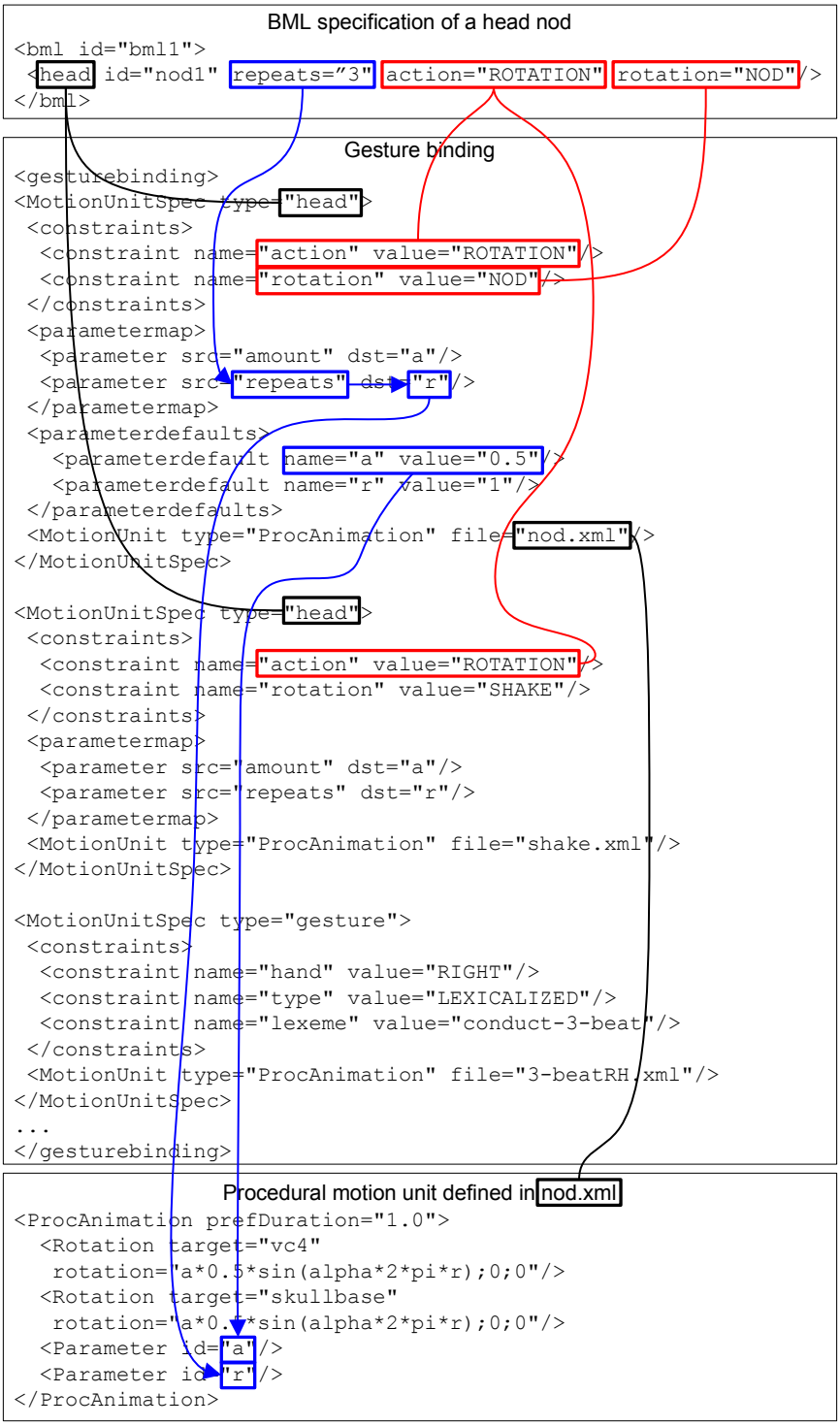


Figure 8.11: GestureBinding fragment binding the <head> element to the nod MotionUnit. Both the nod and shake MotionUnits execute behaviors of type "head". They both satisfy the constraint `action="ROTATION"`, but only the nod MotionUnit satisfies the constraint `rotation="NOD"` and is therefore selected to execute the head nod. The GestureBinding maps the *repeats* parameter value in the BML behavior to the value of parameter r specified in the procedural MotionUnit. The value of parameter a is not defined in the BML head behavior, therefore the default value of a, as defined in the GestureBinding, is used in the procedural animation.

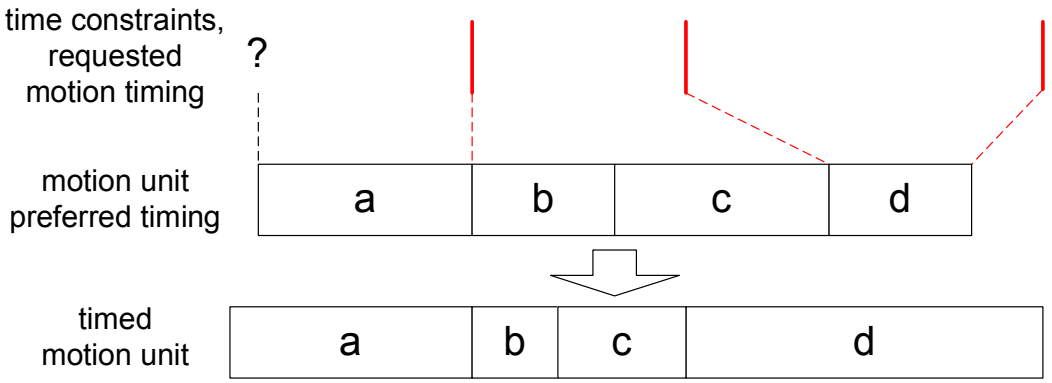


Figure 8.12: The AnimationPlanner is asked to resolve the start time of a behavior with phases a, b, c and d, given 3 constraints on its timing. Phase d is stretched as prescribed by the two rightmost constraints. Uniform scaling is applied to phases b and c to satisfy the two leftmost constraints. Phase a is slightly stretched to maintain the average stretch resulting from the satisfaction of the time constraints on the behavior.

8.7.5 Motion Execution

The PlanPlayer takes an AnimationPlan containing TimedMotionUnits. At any specific point in time, it takes the following steps to animate the virtual human.

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

Mutually exclusive behaviors (e.g. two or more gaze behaviors, or two or more posture behaviors) allow only one active instance at a time. The latest mutually exclusive behavior overwrites previous behavior (see also Chapter 6.3.3). This allows the virtual human to, for example, temporarily divert gaze to a new target and then return gaze to an earlier specified target. Elckerlyc supports this by assigning the same *replacement group* to all MotionUnits corresponding to such behaviors. Within each replacement group, the active timed MotionUnit is defined to be the TimedMotionUnit for which $f_{mi}(0) \leq t < f_{mi}(1)$ and $f_{mi}(0)$ has the largest value. If multiple TimedMotionUnits within the same replacement group share the same value for $f_{mi}(0)$, the TimedMotionUnit with the smallest $f_{mi}(1)$ is selected as the active TimedMotionUnit for the replacement group. If the $f_{mi}(1)$'s are also equal, the TimedMotionUnits are completely overlapping in time. Then, one of the TimedMotionUnits is dropped, and a warning is sent to the SAIBA Behavior Planner.

(2) Determine which physical body configuration should be assigned.
The AnimationEngine infers the continuously changing mix of kinematically and physically steered joints from the active procedural and physical TimedMotionUnits. Each MotionUnit specifies on which joints it acts. The selected physical body configuration is the smallest physical body configuration that contains all joints steered by all active physical TimedMotionUnits. For example, the behavior planner for

the virtual conductor can specify in BML the following behaviors for the left arm: a physical MotionUnit for ten seconds, that lets the arm hang down, followed by a procedural MotionUnit to make a certain expressive conducting gesture for one second, and finally again a physical MotionUnit to let the arm hang down again. To realize this sequence, Elckerlyc automatically executes a switch of the affected joints from physical steering to kinematic steering and back again.

(3) The active procedural TimedMotionUnits are then executed at $\alpha = f_{mi}^{-1}(t)$. Most TimedMotionUnits calculate $f_{mi}^{-1}(t)$ using a simple linear time warping scheme [309]. Others use a more complex warping scheme, for instance one to create a biomechanically inspired bell shaped velocity profile in a pointing gesture [304]. Providing mechanisms to combine procedural motion on the same joint is future work (see also Section 8.17).

(4) The physical simulation is performed as a last step. The generated procedural animation is combined with the active physical TimedMotionUnits, using the method described in Section 8.7.1 (see also Figure 8.9). If different physical controllers act upon the same joint, the sum of their torques is applied to that joint, thus combining their objectives.

8.8 The FaceEngine

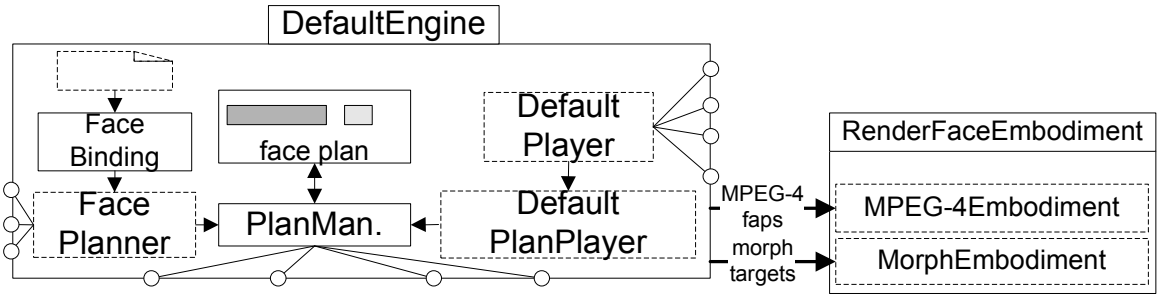
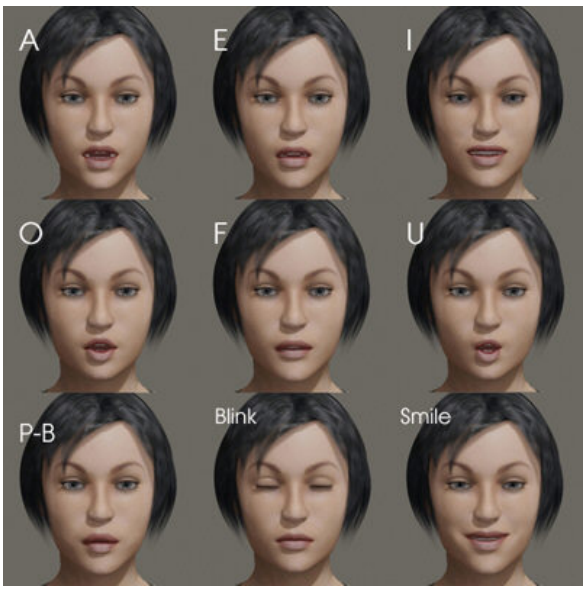


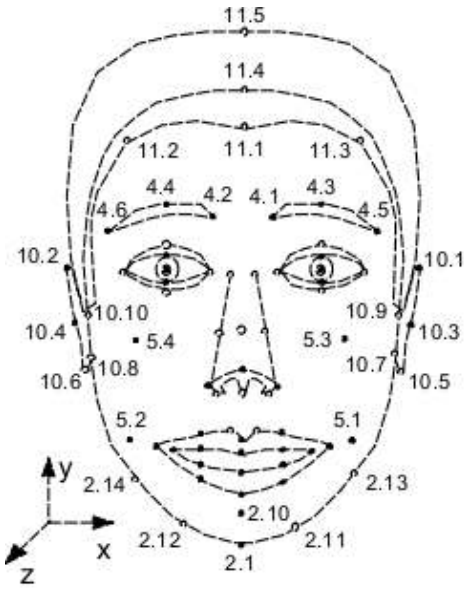
Figure 8.13: The FaceEngine

The FaceEngine (Figure 8.13) manages `face` behaviors. It can combine the naturalness of morph based facial animation with the control of MPEG-4 facial animation by using them simultaneously as control primitives for the RenderFaceEmbodiment.

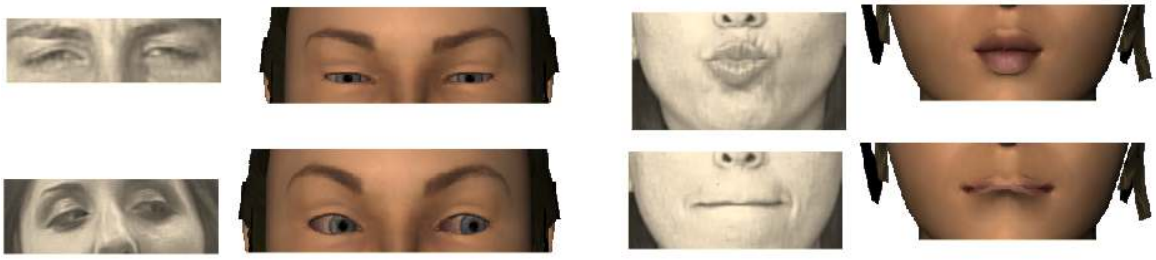
The FaceEngine allows facial animation to be specified using morph targets, MPEG-4 facial action parameters (FAPs), Action Units (AUs) in the Facial Action Coding System (FACS) [71], or using coordinates in Plutchik's emotion wheel [221] (see also Figure 8.14). Morphing uses artist-created morph targets (e.g. for eye blinks and visemes). MPEG-4 animation uses MPEG-4 FAPs: the animation is specified by the position of facial action points on the face. The FaceEngine maps the facial animation descriptions in AUs or emotion wheels coordinates to MPEG-4 FAPs, using mappings from literature [226, 321]. These mappings were implemented by Ronald Paul and their exact implementation is discussed in his master's thesis [216].



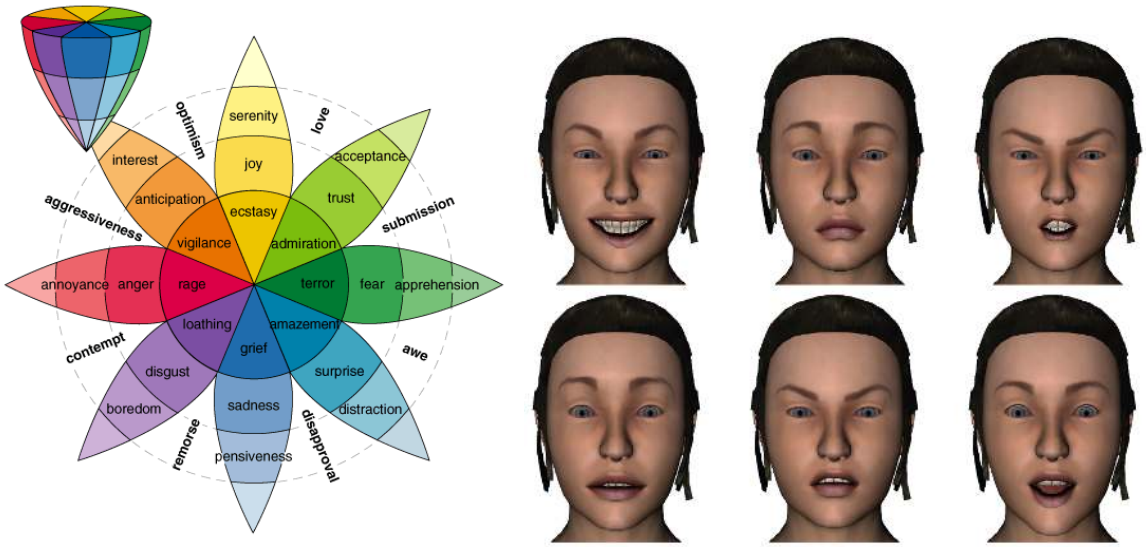
(a) Morph targets.



(b) MPEG-4 feature points.



(c) Some Action Units in FACS and their execution in Elckerlyc.



(d) Plutchik's emotion wheel and some stereotypical emotional face expressions.

Figure 8.14: Facial animation paradigms used in Elckerlyc's FaceEngine

Given certain time constraints on a face behavior, the FacePlanner employs a linear stretching algorithm on its timing, similar to that used to resolve the timing of animation (see Section 8.7.3) to resolve the remaining constraints.

The RenderFaceEmbodiment combines the naturalness of morph-target based animation with the control of MPEG-4 based animation. It implements both the MPEG-4 Embodiment and the MorphEmbodiment, which allows it to capture both types of input and combine them. This combination process first applies the morph specified by the Timed Morph Face Units and then moves the morphed mesh on the basis of the MPEG-4 FAP specification. Alternatively, the FaceEngine can be connected to custom MPEG-4 and/or MorphEmbodiments.

8.9 The Text To Speech Engine

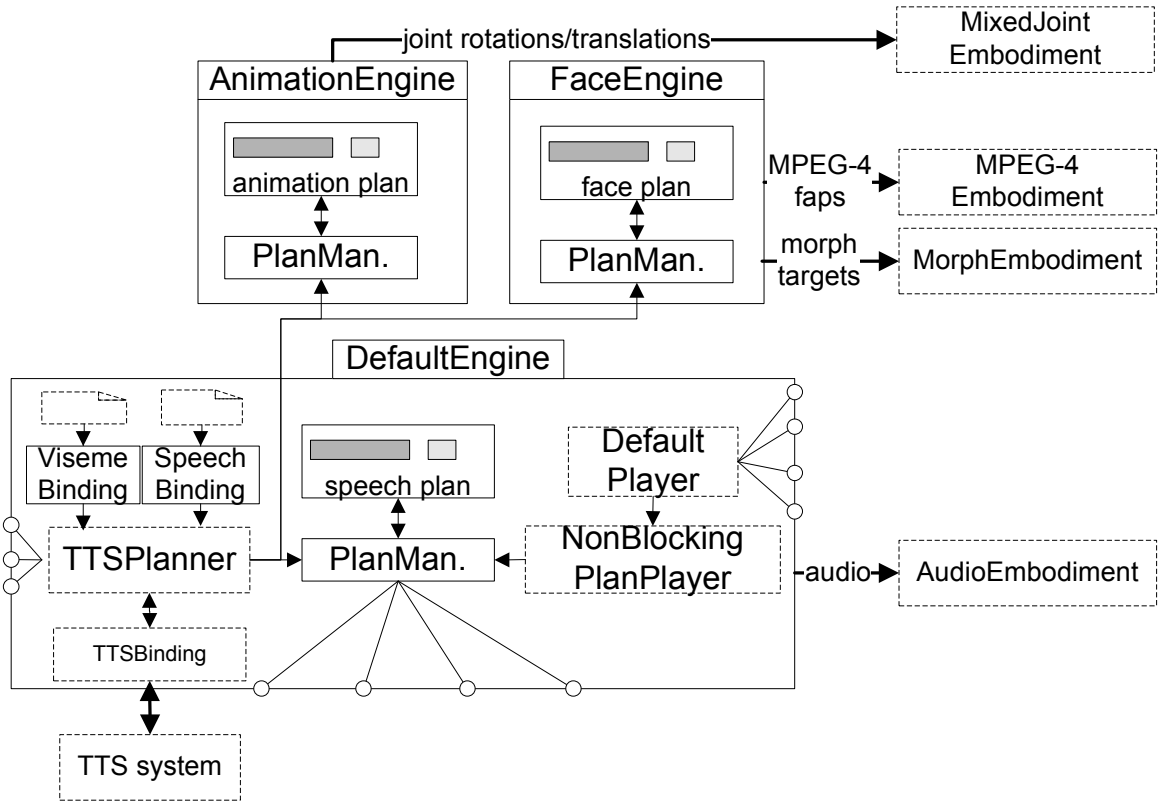


Figure 8.15: The TTSEngine

The Text To Speech (TTS) Engine (Figure 8.15) manages `speech` behaviors. It executes these behaviors using a TTS system. The TTSBinding provides a generic interface for such TTS systems. Current implementations include a MaryTTSBinding (using Mary TTS[12]) and a SAPITTSBinding (using Microsoft Speech API[13]). The TTSPlanner generates TimedTTSUnits from a speech behavior specification. The

[12] `http://mary.dfki.de/`
[13] `http://www.microsoft.com/speech`

TimedTTSUnits specify the sounds to be spoken. In addition to TimedTTSUnits, a TTSPlanner may generate TimedFaceUnits and TimedMotionUnits that specify the speech lip and/or jaw movement respectively. The TTS Engine requires access to the PlanManager of the AnimationEngine and the PlanManager of the FaceEngine to add these units to their respective Plans. The VisemeBinding and SpeechBinding define the mapping from visemes (facial phonemes) to FaceUnits or MotionUnits respectively. At initialization time, the TTSEngine is configured with the desired VisimeBinding, SpeechBinding and TTSBinding.

The TTSPlanner uses the TTS system (through the interface provided by its TTSSpeechBinding) to determine the timing of the requested speech fragment, including the timing of any custom sync points in it. The internal timing of TimedTTSUnits constructed by the TTSPlanner is fixed; that is, the TTSEngine does not support stretching or skewing of speech beyond the timing provided by the TTSBinding.

8.10 The TextEngine

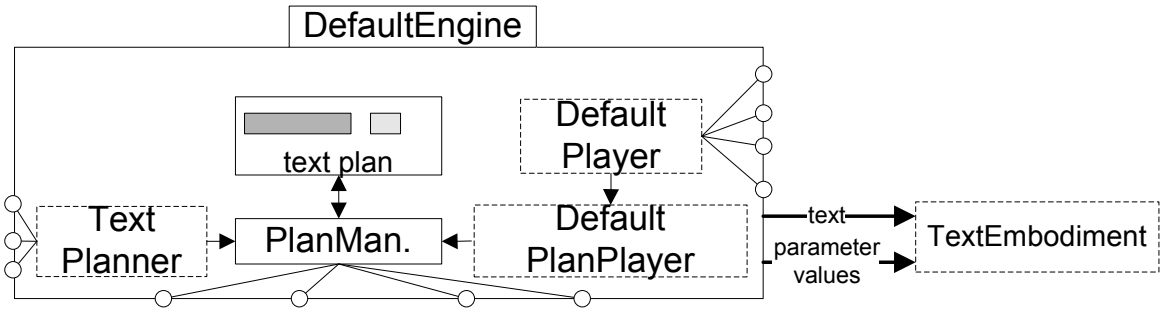


Figure 8.16: The TextEngine

The TextEngine (Figure 8.16) was designed as an alternative to the TTSEngine that allows a more flexible timing of `speech` behaviors. The TextPlanner constructs TimedTextUnits from a speech behavior specification. A TimedTextUnit 'speaks' by sending the text to be said incrementally (by default at a rate of two words per second) through a TextEmbodiment. The TextEmbodiment is a generic interface for elements that can receive text and optionally can handle speech parameters such as amplitude. The desired TextEmbodiment implementation is set connected to the TextEngine at initialization time. Current implementations include a JLabelTextEmbodiment that displays the text on a Java GUI interface element and a StdoutTextEmbodiment that displays the text on the standard output. The JLabelTextEmbodiment implements the speech amplitude parameter by mapping it to font size.

The TextPlanner allows the timing of a speech behavior to be adapted to multiple provided time constraints (e.g. from other behaviors, Anticipators). It employs a linear stretching algorithm to resolve the time constraints. This algorithm is similar to that in used to resolve the timing of animation (see Section 8.7.3).

8.11 The AudioEngine

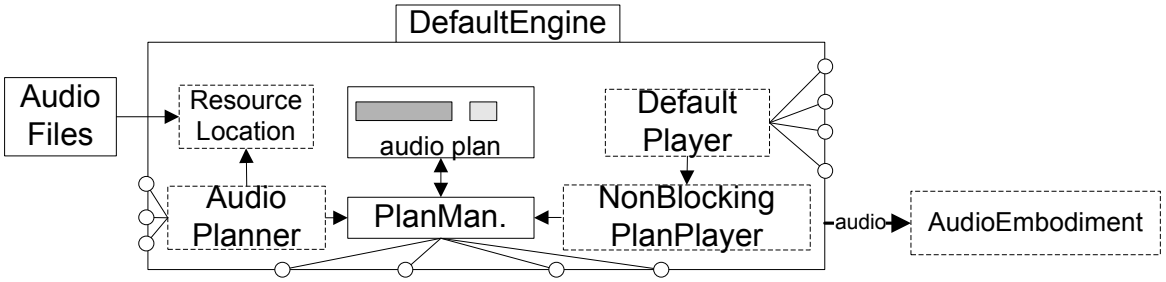


Figure 8.17: The Audio Engine

The AudioEngine manages BML^{T} `audiofile` behaviors. Audio file behaviors are used to play audio from a file. The AudioEngine is implemented using mostly 'stock' components (see Figure 8.17). A specialized AudioPlanner constructs TimedAudioUnits from the audiofile behavior specification. It is provided with a audio file resource location (e.g. a directory, or a .jar file) at initialization time. It can open the audio files specified in the behavior at the resource location.

The audio file behavior currently has only two valid sync points: its start and its end. The audio file defines the duration of the behavior, the AudioPlanner currently disallows squeezing or stretching of TimedAudioUnits.

8.12 The WaitEngine

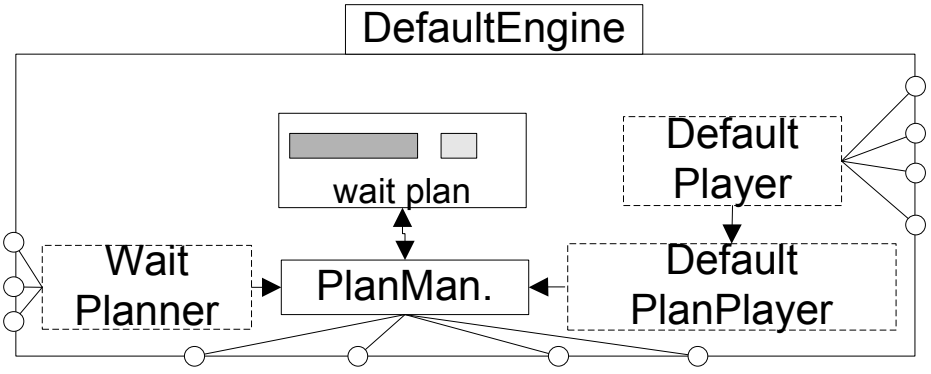


Figure 8.18: The WaitEngine

The WaitEngine manages `wait` behaviors. Wait behaviors are used both as a simple no-operation behavior to offset the timing of other behaviors and in an event based synchronization system that is to be used to synchronize inter-virtual human behavior. Elckerlyc currently only supports the first.[14] The WaitEngine is implemented using mostly 'stock' components (see Figure 8.18).

[14]The specification of the latter is still an open issue in BML.

The wait behavior has two relevant sync points: its start and its end. A specialized WaitPlanner constructs a TimedWaitUnit from a wait behavior and resolves its timing. Typically it is asked to provide the timing of one of the wait's sync points, given the timing of the other and the wait's duration.

8.13 The ActivateEngine

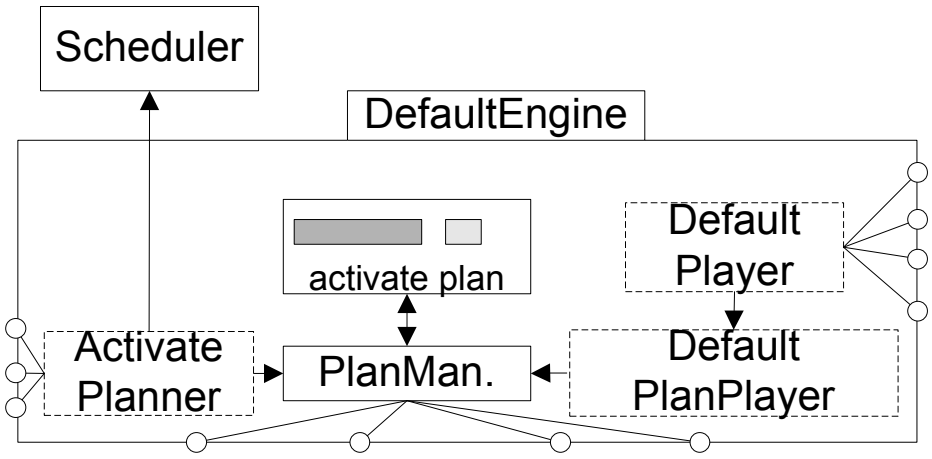


Figure 8.19: The ActivateEngine

The ActivateEngine (Figure 8.19) manages BML^{T} `activate` behaviors. Activate behaviors are used to activate their target pre-planned BML block.

The ActivatePlanner constructs a TimedActivateUnit based on the activate behavior. It provides this TimedActivateUnit with access to the Scheduler, so that it can use Scheduler functionality to activate the targeted pre-planned BML block. The activate behavior may only be synchronized at its start sync point.

The ActivateEngine executes TimedActivateUnits as soon as the current time is past their start time. When they are first executed, TimedActivateUnits activate their target pre-planned BML block. After activating this block, the TimedActivateUnit is finished.

8.14 The InterruptEngine

The InterruptEngine manages BML^{T} `interrupt` behaviors. Interrupt behaviors are used to interrupt a list of targeted behaviors, as specified by the target BML Block and the behavior's include and exclude attributes. The InterruptEngine is implemented using mostly 'stock' components (see Figure 8.20).

The InterruptPlanner constructs an TimedInterruptUnit based on the interrupt behavior. It provides this TimedInterruptUnit with access to the Scheduler, so that it can use Scheduler functionality to interrupt the targeted behaviors. The interrupt behavior may only be synchronized at its start sync point.

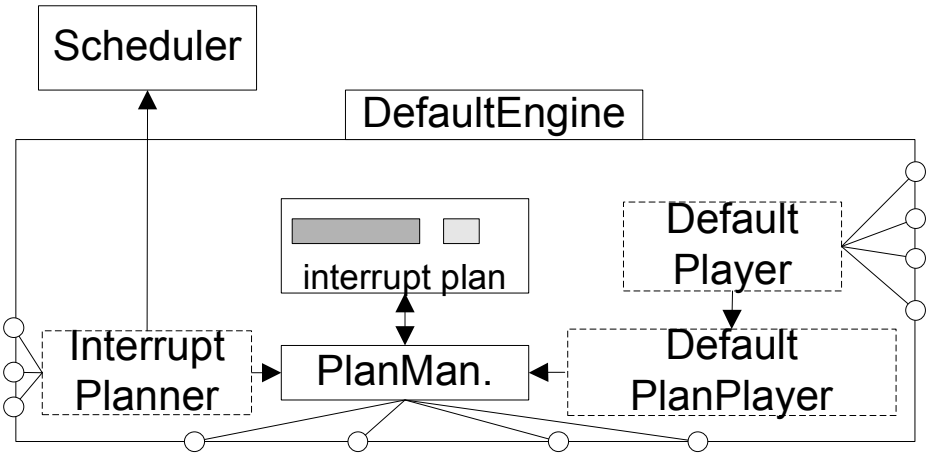


Figure 8.20: The InterruptEngine

The InterruptEngine executes TimedInterruptUnits as soon as the current time is past their start time. At the start of the TimedInterruptUnit it interrupts all its targeted behaviors. A TimedInterruptUnit makes use of the Scheduler to interrupt its targeted behaviors.

8.15 The ParameterValueChangeEngine

The ParameterValueChangeEngine manages BML[T] `parametervaluechange` behaviors. A parameter value change behavior is used to change parameter values in another behavior (for example: change the volume in a speech behavior). A parameter value change behavior contains two sync points: a start and a stroke. The stroke sync point indicates when the target parameter value is to be achieved. The ParameterValueChangeEngine is implemented using mostly 'stock' components (see Figure 8.21).

A ParameterValueChangePlanner constructs a TimedParameterValueChangeUnit. The TimedParameterValueChangeUnit is assigned a trajectory based on the trajectory type specified in its behavior. New trajectories can be designed by adding them to the TrajectoryBinding. The current TrajectoryBinding contains a trajectory that moves from start to end parameter value in a linear curve and one that sets the

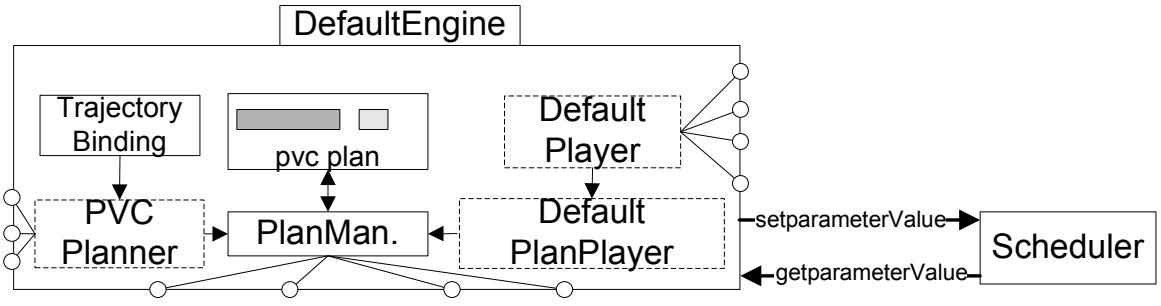


Figure 8.21: The ParameterValueChangeEngine

end parameter value instantly. The TimedParameterValueChangeUnit is provided with access to the Scheduler, so that it can set and get the parameter value of other behaviors in the behavior plan.

The Player executes a TimedParameterValueChangeUnit as soon as the current time is past its start time. The trajectory provides a parameter value at canonical time α:

$$\alpha = \begin{cases} \frac{t - t_{start}}{t_{stroke} - t_{start}} & \text{if} \quad t_{stroke} > t_{start} \quad \text{and} \quad t < t_{stroke} \\ 1 & \text{otherwise} \end{cases} \tag{8.1}$$

where t is the current time and t_{start} and t_{stroke} are the start and stroke times of the behavior respectively. Given α, an initial parameter value v_i and a target parameter value v_t, the trajectory function $f(v_i, v_t, \alpha)$ determines the parameter value at canonical time α. Equation 8.2 and equation 8.3 show the trajectory function for an instant and a linear trajectory, respectively.

$$f(v_i, v_t, \alpha) = v_t \tag{8.2}$$

$$f(v_i, v_t, \alpha) = v_i + \alpha(v_t - v_i) \tag{8.3}$$

This calculated parameter value is set through the Scheduler at each execution frame. If the initial value of a parameter is not set in the behavior, it is obtained from the Scheduler at the start of the TimedParameterValueChangeUnit.

8.16 Extending and Using Elckerlyc

8.16.1 Adding New BML Elements and PlanUnits

Elckerlyc offers a large repertoire of PlanUnit types, in various Engines, that can be mapped in a Binding to give form to the abstract BML behaviors: physical simulation, procedural animation, morph target and MPEG-4 face control, speech, and so on. Still, a developer may need completely new PlanUnit types. For example, to make the virtual human more lively, one may want to add a PerlinNoiseUnit that applies random noise to a certain joint of the virtual human, as a kind of 'idle motion'. Such new PlanUnits need to become available in the Binding (see previous section); furthermore, one might want to extend the XML format of BML with `<PerlinNoiseBehavior>` to allow direct specification of this idle motion by the SAIBA Behavior Planner (see Figure 8.22 for an example BML specification of a Perlin noise behavior).

New BML behaviors are created by sub-classing the abstract class BMLBehaviorElement; they can be registered with the Parser using a static call. At initialization of Elckerlyc, the new BML behavior type is coupled to a single Engine by adding it to the behavior class $\rightarrow$ engine mapping (note that multiple behavior types can be coupled to the same Engine).

New PlanUnits implement the PlanUnit interface (for the AnimationEngine: rotate joints on the basis of time and animation parameters, see also Chapter 4.2). Such PlanUnits are initialized from the GestureBinding through their class name (as

```
<bml id="bml1" xmlns:mynoise="http://mynoise.net">
  <mynoise:perlinnoise id="n1" joint="vc4"
    baseamplitudex="0.1" basefreqx="1"/>
  ...
</bml>
```

Figure 8.22: An XML block with a custom noise behavior. Note that the noise behavior has its own namespace.

```
<gesturebinding>
  ...
  <MotionUnitSpec type="perlinnoise" namespace="http://mynoise.net">
    <parametermap>
      <parameter src="joint" dst="joint"/>
      <parameter src="baseamplitudex" dst="baseamplitudex"/>
      <parameter src="baseamplitudey" dst="baseamplitudey"/>
      <parameter src="baseamplitudez" dst="baseamplitudez"/>
      <parameter src="basefreqx" dst="basefreqx"/>
      <parameter src="basefreqy" dst="basefreqy"/>
      <parameter src="basefreqz" dst="basefreqz"/>
      ...
    </parametermap>
    <MotionUnit type="class"
      class="mynoisepackage.PerlinNoisePU"/>
  </MotionUnitSpec>
</gesturebinding>
```

Figure 8.23: Binding a custom Perlin noise behavior to a custom noise PlanUnit.

a string), using Java's reflection mechanism (that is, the ability to construct a new object from its class name). This ensures that any PlanUnit implementing the right interface for an Engine can be used in the Binding for that Engine without requiring additional compile time dependencies. Figure 8.23 illustrates this. The `noise` behavior in namespace `http://mynoise.net` is bound to the PerlinNoiseUnit in the Java class `mynoisepackage.PerlinNoisePU`.

8.16.2 Adding a New Engine

Building a new Engine involves developing its basic PlanUnits, that implement the basic control primitives for the modality. Each PlanUnit defines how it will control an Embodiment over the duration from the start time till the end of the PlanUnit. Given these PlanUnits, and a Binding for mapping BML behaviors to PlanUnits, a new Engine is typically constructed using the standard available Engine components. An example is provided in Chapter 10.1.3.2: it outlines how a new Engine was developed to steer a Nabaztag robot rabbit with BML.

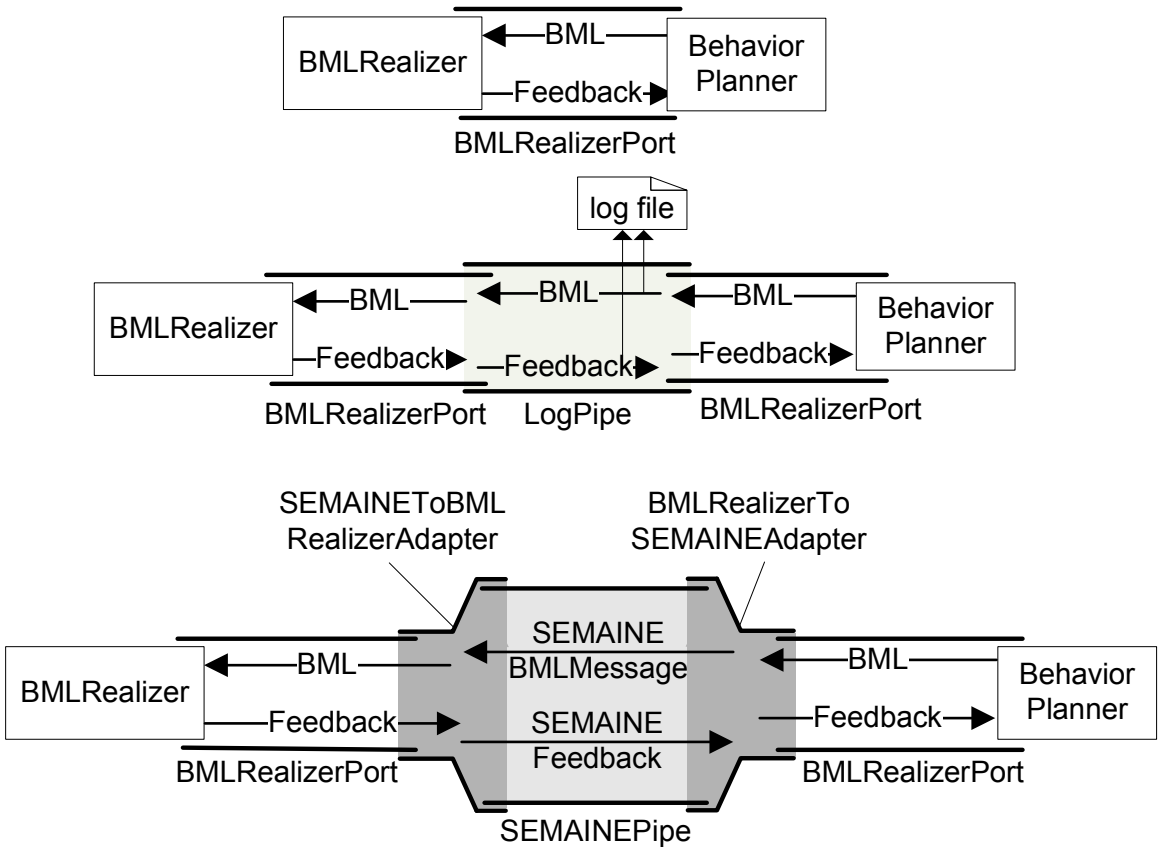


Figure 8.24: Top: the BML Realizer and SAIBA Behavior Planner are connected directly on a Realizer port. Bottom: the BML Realizer and SAIBA Behavior Planner are connected through the SEMAINE API; the BML Realizer and SAIBA Behavior Planner are unaware of this plumbing, they still SAIBA communicate through BMLRealizerPorts. Middle: a LogPipe logs the BML and feedback messages that pass through it to a file.

8.16.3 Connecting Elckerlyc to Your Application: Ports, Pipes, and Adapters

A minimal interface to a BML Realizer has functionality to (1) send a BML string to the Realizer and (2) register a listener for Realizer feedback. This is the BMLRealizerPort in Figure 8.6. Both the SAIBA Behavior Planner and the BML Realizer are connected to such a BMLRealizerPort. The adapter pattern [83] allows one to change the exact transport of BML and feedback to and from a BML Realizer, with no impact on the SAIBA Behavior Planner and BML Realizer.

Elckerlyc implements the BMLRealizerPort interface. Several adapters were implemented that plug into BMLRealizerPorts and transport their messages over various messaging frameworks. Pipes are used to intercept BML and feedback, allowing one to measure it, let it go through slightly modified, or at a different rate. For example, a LogPipe logs the BML and feedback passing through, and a BlockingQueuePipe buffers BML messages for a BMLRealizerPort that can only handle one BML message at a time. Figure 8.24 shows some examples.

8.16.4 Integrating Elckerlyc with new Embodiments

Engines set control primitive values on their Embodiments. Each Engine uses a specific set of Embodiments. For example, the FaceEngine uses an MPEG-4Embodiment and a MorphEmbodiment. New Embodiments that implement an existing Embodiment interface can be coupled to an Engine at the Engine's initialization time. For example, rather than coupling the FaceEngine to its default MPEG4-Embodiment, it can be coupled to an MPEG-4 Embodiment that steers the XFace [15] talking head. The next Section discusses a specialized SkeletonEmbodiment that was designed to set the joint rotations provided by the AnimationEngine to a virtual human in any Rendering Environment.

8.16.5 Integrating Elckerlyc with new Rendering Environments

By default, Elckerlyc renders the virtual human in its own OpenGL based rendering environment. One might, however, want to use Elckerlyc to animate an Embodiment in another rendering environment such as Half-Life, Ogre, or Blender. The ThriftSkeletonEmbodiment was developed to integrate a SkeletonEmbodiment with *any* desired rendering environment.

To provide a clear seperation between the virtual human in the rendering environment and the SkeletonEmbodiment that is animated by Elckerlyc, we follow a design similar to that proposed by Russel and Blumberg [245]. The AnimationEngine animates a local copy of the the SkeletonEmbodiment. The joint rotations set by Elckerlyc are copied to the virtual human in the rendering environment regularly (typically each frame) by the ThriftSkeletonEmbodiment.

The rendering environment therefore needs to support functionality to (1) provide the ThriftSkeletonEmbodiment with the joint structure of the virtual human at its initialization, and (2) provide the ThriftSkeletonEmbodiment with means to copy joint rotations to the virtual human in the rendering environment. Both requirements should be satisfied in a manner independent of the specific rendering environment used and of the mode of transport (e.g. through TCP/IP, function call, shared memory). I use the remote procedural call framework Thrift [272] to achieve this. I have designed a language independent interface (using Thrift's interface definition language) that a rendering environment should implement to achieve connectivity with a ThriftSkeletonEmbodiment. This interface is automatically compiled to an interface in the target language of the rendering environment. The transport mode is chosen at initialization time. A proof-of-concept implementation is provided for the Ogre rendering environment.

8.17 Discussion and Future Work

I have presented Elckerlyc, a modular and extensible BML Realizer designed specifically for continuous interaction. It offers an adjustable trade-off between the control offered by procedural animation and the naturalness enhancements offered by

physical simulation. I have shown how these capabilities work in practice in the interactive virtual conductor using a running example. I have assumed that Elckerlyc's Engines can operate independently from each other. In practice, some links between Engines might be required, sacrificing some independency for performance or time synchronization reasons. This is a common trade-off in the design of Virtual Human platforms [285]. One example of such a dependency is in the TTSEngine. This Engine requires access to the AnimationEngine and the FaceEngine to add some PlanUnits that execute the animation corresponding with the speech it manages. Currently all Engines execute their plans timed by the same global clock. The timing of off-the-shelf text-to-speech system can drift significantly from the the global clock. One implementation of the TTSEngine uses the text to speech system via an intermediate audio file and enforces time constraints by skipping forward and backward in the audio file on significant time drifts (more than 70ms). This introduces some sound artifacts and still suffers from minor timing differences between speech and lip-sync. Finer synchrony and better speech quality could be achieved by allowing the SpeechEngine to run at its own timing and communicate the drift of this timing in relation to the global clock to the other Engines using TimePegs. Such a design makes elegant use of Elckerlyc's generic synchronization mechanism: the TimePegs are used to communicate synchronization constraints between Engines.

I have discussed how Elckerlyc can be tailored to the needs of specific applications, without requiring invasive modifications to Elckerlyc itself. Elckerlyc's flexibility has allowed its connection to a SAIBA Behavior Planner using either the SEMAINE framework or simple function calls, and to switch between such connections with a simple configuration option. The logging port allowed easy recording of all communication with Elckerlyc for user experiments, by simply changing the wiring between the SAIBA Behavior Planner and Elckerlyc. The BMLRealizerPort also made exchanging both the Realizer and the SAIBA Behavior Planner very easy. Several SAIBA Behavior Planners were designed at HMI that implement behavior planning of a virtual human or that replace the behavior planning by a generic Wizard of Oz interface (see Chapter 10.4.1).

The ability to easily replace the BML Realizer and SAIBA Behavior Planner is also valuable for testing. A mock-up BML Realizer was designed that allows the testing of SAIBA Behavior Planners rapidly. This mock-up BML Realizer does not actually execute the BML behavior, but does provide the SAIBA Behavior Planner with appropriate BML feedback. I have also designed a SAIBA Behavior Planner that tests Realizer implementations (see Chapter 9.3). This SAIBA Behavior Planner executes test BML scripts on the Realizer and inspects if the Realizer provides the appropriate feedback. Since this test SAIBA Behavior Planner communicates with the Realizer through the generic BMLRealizerPort, it can not only test any configuration of Elckerlyc, but can also test Realizers designed by other research groups (by writing an adaptor from the BMLRealizerPort to their input and output channels). Elckerlyc's ability to add new modalities has allowed hooking it up to the Nabaztag rabbit (See also Chapter 10.1.3.2) and to steer this rabbit with generic BML commands.

Elckerlyc's extensibility is mainly achieved by a very flexible initialization stage.

In this initialization stage, a desired setup of the Elckerlyc Realizer is constructed by combining and configuring different components that are provided by Elckerlyc's code base or by custom extensions. An XML configuration file format is used to describe such a configuration. Several default configurations are available, and new configurations are typically easily achieved by slight modifications of an existing configuration.

An Elckerlyc demo application (as Java webstart) and several demonstration movies and BML scripts can be found on the web page of this thesis.

An important topic for future work is *conflict resolution*. Several behaviors might request the use of the same animation modality, for example the right arm or head. Elckerlyc does not currently resolve such conflicting modality requirements. A simple custom conflict solver is used in the SAIBA Behavior Planner of the virtual conductor application. Conflict resolution requires resource allocation (e.g. select the left arm for a gesture if the right arm is doing something else) and resource combination (for instance: combine a gaze with a nod). Resource allocation could be achieved in Elckerlyc by taking into account the currently used resources when selecting a MotionUnit from the GestureBinding. A MotionUnit can store information on how it might be combined with other MotionUnits [205]. For example, a nod motion should be additively added on top of other head movements [280] and a manipulatory gesture should constrain hand position [111], but might allow elbow movements. The Animation Planner could construct (possibly hierarchical) combinatory MotionUnits that solve resource conflicts [280]. This way, the Animation Plan only contains non-conflicting MotionUnits. Alternatively, the AnimationPlayer could use a final phase that combines conflicting MotionUnits [111]. I have given an overview of motion combination techniques for both physical and kinematic motions in Chapter 2.

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

Currently the GestureBinding provides a one-to-one mapping from a BML behavior to a MotionUnit. Extending the GestureBinding to allow one-to-many mappings might be useful to select the best MotionUnit on the basis of available modalities at its execution time (left hand, right hand) or on the basis of its time constraints (select the MotionUnit with the shortest preferred duration if fast execution is required) or just to select a random MotionUnit for, for example, a beat gesture.

Chapter 9

Demonstrating and Testing the BML Compliance of BML Realizers†

The SAIBA framework [152] has standardized the architecture of virtual human applications with the aim of making reuse of their software components possible. The SAIBA framework proposes a modular 'planning pipeline' for real-time multimodal motor behavior of virtual humans, with standardized interfaces (using representation languages) between the modules in the pipeline. One of the components in this pipeline is the Realizer. A Realizer provides an interface to steer the motor behavior of a virtual human: a description of behavior in the Behavior Markup Language (BML) goes 'in', feedback comes 'out'.

Several Realizers have been implemented ([54, 111, 186, 280] and Elckerlyc). If SAIBA's goal of software reuse is achieved, it will be possible to use such Realizers interchangeably with the same BML input. This chapter discusses a collaborative effort by the authors of SmartBody[280] and Elckerlyc Realizers. We are interested in measuring and promoting the compatibility between our own and other Realizers and to provide tools to formally test and maintain adherence to the BML standard. To this end, we provide a growing test set of BML test cases, a corpus of BML scripts and video material of their realization in different (so far, our two) Realizers.

By directly comparing BML Realizers, we can better determine changes to the BML specification that are necessary due to overly narrow or broad specifications. Overly broad specifications can be detected when Realizers provide BML compliant, but semantically very different results. Overly narrow specifications indicate that a specification is not expressive enough, this occurs when several Realizers implement the same (semantic) functionality, yet require Realizer specific proprietary BML extensions to implement (part of) this functionality.

Since each Realizer necessarily implements the same interface, an automatic testing framework can be designed that tests the adherence to the BML/feedback

†This chapter is largely based upon the article:
H. van Welbergen, Y. Xu, M. Thiebaux, W.-W. Feng, J. Fu, D. Reidsma, A. Shapiro. Demonstrating and Testing the BML Compliance of BML Realizers, In Proceedings of the 11th International Conference on Intelligent Virtual Agents, 2011, To appear

semantics for *any* Realizer. We contribute our testing framework RealizerTester[1], which provides exactly this functionality.

9.1 On BML Versions and Script Creation

Currently, there are two version of the BML specification: a first draft specified after the BML workshop in Vienna in November 2006 (the Vienna draft) and the current draft of BML version 1.0 (draft 1.0). Draft 1.0 is not backward compatible with the Vienna draft. Currently Elckerlyc implements draft 1.0, and SmartBody implements the Vienna draft. It is likely that new versions of BML will be developed and that not all Realizers will adapt to these new versions at the same pace. However, test scripts can be constructed that are semantically equivalent (that is, execute behavior that adheres to the same form and timing constraints) for most, if not all, BML behaviors in different versions of BML. This implies that the same test case (albeit not test script) can be used to test Realizers that implement different versions of BML.

We aim to construct a test set that contains such semantically equivalent test cases for all BML versions. These tests will provide a 'safety net' for migrating a Realizer from a previous BML version to the next; the tests that worked for a Realizer in the previous version should not break in their updated syntax in a later version of BML.

The process of converting the old tests to a new BML version also helps in the definition of the standard. It can highlight certain cases in which expressivity is lost where this might not be intended. That is: if something can be expressed in a previous version of BML which we cannot express in the new version of BML and this loss of expressivity was not explicitly intended in the new BML version, then there might be something 'wrong' in the definition of the new version.

Most of our current test scripts were originally designed for draft 1.0 and later converted to equivalent Vienna draft scripts. During this conversion process, we have encountered several cases that demonstrate the enhanced expressivity of the newer draft 1.0. For example:

- In the Vienna draft it was impossible to specify a Realizer-independent posture behavior; draft 1.0 provides the specification of some default lexicalized postures.
- Draft 1.0 provides the specification of a modality (e.g. eyes, neck, torso) in gaze behaviors. The Vienna draft does not allow this. Therefore, SmartBody currently needs to use a custom extension to specify the modality of its gaze behavior.

A small set of scripts was converted from a Vienna draft specification to a draft 1.0 specification. So far we have not encountered any cases in which expressivity of the Vienna draft was lost in draft 1.0.

[1] RealizerTester is released under the MIT license at `http://sourceforge.net/projects/realizertester/`.

9.2 A Corpus of Test Cases and Videos

We provide a growing corpus of BML scripts for the purpose of visual comparison between the execution of a BML script by the different Realizers. A matching video corpus illustrates how these scripts are executed in both SmartBody and Elckerlyc. The corpus of BML scripts (and matching video) provides examples of both short monologues and of the execution of isolated behaviors. A video on the web site of this thesis and Figure 9.1 illustrate the comparison of the execution of some of these scripts in SmartBody and Elckerlyc. Here we discuss some preliminary observations.

Script are included for most types of BML behavior. In the scripts used so far, the `speech`, `gaze`, `head` and pointing (part of `gesture`) behaviors look similar in Elckerlyc and SmartBody. A comparison of `face` behaviors, specified through Ekman's FACS [71] gives mixed results, as illustrated in the video. The `posture` behavior could not be compared in a meaningful manner, because the Vienna BML draft does not provide a Realizer independent way to specify posture. The `locomotion` behavior is currently not implemented in Elckerlyc and is therefore omitted from the visual comparison corpus for now.

When designing short monologues for the visual comparison corpus, it became clear that existing demonstration scripts of both SmartBody and Elckerlyc rely heavily upon custom behavior elements. One reason for this is the lack of expressivity of the BML standard and specifically the lack of expressivity in the specification of iconic and metaphoric gestures. We recommend, at the very least, the extension of the lexicalized set of gestures that can be specified in BML to include more domain neutral gestures. The set of gestures that is already implemented through extensions by current Realizers could serve as an inspiration for this. Another reason for the use of custom BML elements is that so far there was no real need for BML compliance of Realizers. Some BML elements that are currently implemented using one or more custom BML extensions in Elckerlyc and SmartBody could be implemented in standard BML. This testing and comparison corpus building effort serves as a driving force for this. Already some new core BML behaviors have been implemented in both Realizers to achieve better BML compliance.

We have created a test corpus containing 19 test scripts in BML draft 1.0, semantically equivalent test scripts in the Vienna draft, and the corresponding test cases that check, for example, the adherence to the time constraints specified in the scripts.

9.3 Automatic Software Testing of Realizers

Realizers are complex software components. They often form the backbone of several virtual human applications of a research group. Therefore, the stability and extendibility of Realizers is crucial. So far, the testing of most of these Realizers was limited to *manual*, time consuming inspections of the execution of a selected

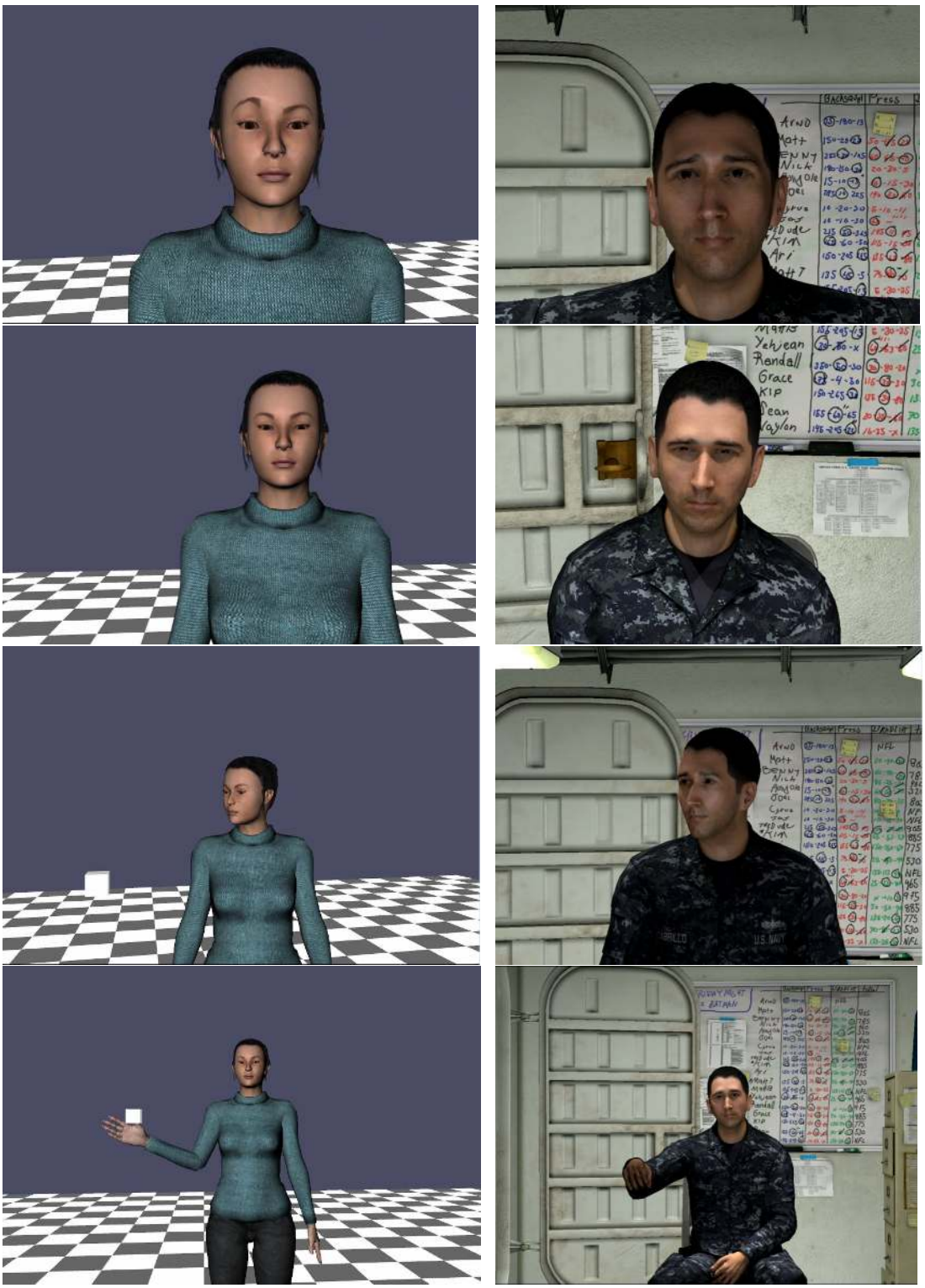


Figure 9.1: Execution of some BML behaviors in Elckerlyc (left) and SmartBody(right). From top to bottom: AU 1 (inner eyebrow raise), AU 6 (cheek raiser and lid compressor), gaze, point.

set of BML scripts.[2] *Automatic* testing can be used to detect errors in Realizers and provides a 'safety net' that can, to some extent, ensure that extensions or design cleanups do not introduce a failure in existing functionality. Since the automatic tests do not require manual intervention, they can be run often which ensures that errors are detected early, which makes it easier to fix them.

We illustrate the use of RealizerTester by describing how it was integrated in the software development process of the Elckerlyc Realizer and discuss how it can contribute to the emerging BML standard.

A SAIBA Behavior Planner communicates with a Realizer by sending BML blocks with intended behavior to it and by capturing the feedback provided by the Realizer. A BML block defines the form and relative timing of the behavior that a Realizer should display on the embodiment of a virtual human. The Realizer is expected to provide the SAIBA Behavior Planner with feedback on the current state of the BML blocks it is executing: it notifies the SAIBA Behavior Planner of the start and stop of each BML block (performance start/stop feedback) and on the passing of sync points for each behavior in the block (sync-point progress feedback). Execution failures are sent using warning and exception feedback.

RealizerTester acts as a SAIBA Behavior Planner: it sends BML blocks to the Realizer Under Test (RUT)[3] and verifies whether the feedback received from the RUT satisfies the assertions implied by the BML blocks. This allows automatic testing of the following properties:

1. *Message Flow and Behavior Execution*: RealizerTester can verify whether the performance start/stop of each BML block and sync-point progress feedback messages of each behavior was received in the correct order and only once. This implicitly provides some information on whether or not the behaviors were actually executed.

2. *Time Constraint Adherence*: a BML block defines several time constraints upon its behaviors. It can require that a sync point in one behavior occurs simultaneously with a sync point in another behavior, or that a certain sync point should occur before or after another one. These constraints can be tested by inspecting the sync-point progress feedback.

3. *Error Handling*: The error handling of a Realizer can be tested by inspecting its warning and exception feedback. For example, RealizerTester can send BML blocks to the Realizer that are invalid or impossible to schedule and then check if the Realizer generated the appropriate exception feedback.

9.3.1 Test Architecture

Each automatic test consists of four phases [193]:

[2] Personal communication with the authors of SmartBody, RealActor, EMBR, Greta and MARC, 2010.

[3] after System Under Test used in [193]

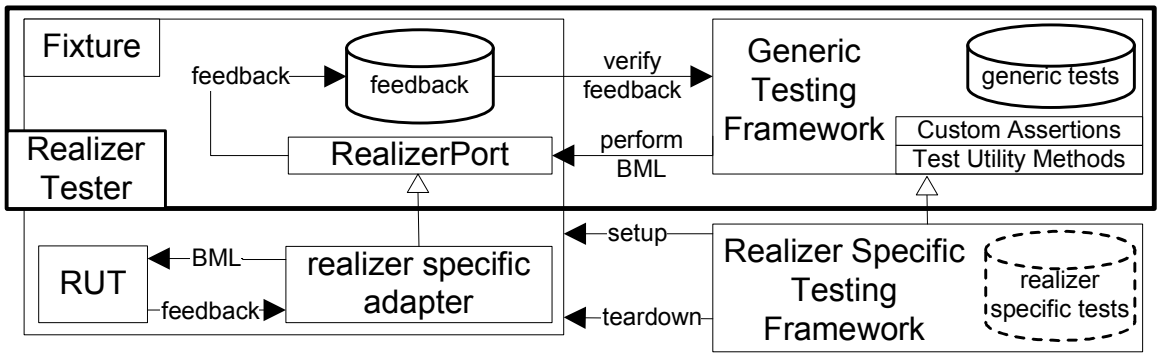


Figure 9.2: Testing Architecture

1. *Fixture setup*: The Fixture contains the RUT and everything it depends on to run. During Fixture set up, the RUT is created and put in a state suitable for testing. The necessary functionality to keep track of the feedback sent by the RUT is also hooked up.

2. *Exercise the RUT*: Send BML block(s) to the RUT.

3. *Result verification*: Verify the feedback received from the RUT.

4. *Fixture teardown*: Clean up the Fixture.

Phases 1 and 4 are Realizer specific, phases 2 and 3 are generic. The Generic (Realizer independent) Testing Framework contains a set of tests and is responsible for exercising these tests and verifying their results. The Generic Testing Framework exercises the RUT by communicating with it through a RealizerPort (see also Chapter 8.16.3). The RealizerPort is a minimal interface for a Realizer.

A Realizer Specific Testing Framework is responsible for setting up and tearing down the Fixture before and after each test case. During the setup phase, this framework creates a Realizer specific implementation of the RealizerPort and connects it to the RUT. RealizerTester is implemented using the JUnit[4] unit testing framework. Since the Fixture setup and teardown is the same for each test, they are implemented using Implicit Setup and Teardown [193] patterns (available in JUnit): setup and teardown functions are called automatically before and after each test respectively. Figure 9.2 shows our architecture setup.

9.3.2 Authoring Test Cases

Test cases are typically set up as follows:

1. Send one or more BML blocks to the RUT, capture all feedback.

2. Wait until the RUT has finished executing all blocks.

3. Verify some assertions on the received feedback.

[4]http://www.junit.org/

```
<bml id="bml1">
    <speech id="speech1" start="6">
        <text>Hey punk <sync id="s1" />what do ya want?</text>
    </speech>
    <head id="nod1" action="ROTATION" rotation="X" start="speech1:s1"/>
</bml>

@Test public void testSpeechNodTimedToSync() {
    realizerPort.performBML(readTestFile("speech_nodtimedtosync.xml"));
    waitForBMLEndFeedback("bml1");
    assertSyncsInOrder("bml1", "speech1", "start", "ready", "stroke_start",
        "stroke", "s1", "stroke_end", "relax", "end");
    assertAllBMLSyncsInBMLOrder("bml1", "nod1");
    assertBlockStartAndStopFeedbacks("bml1");
    assertRelativeSyncTime("bml1", "speech1", "start", 6);
    assertLinkedSyncs("bml1", "speech1", "s1", "bml1", "nod1", "start");
    assertNoExceptions();
    assertNoWarnings();
}
```

Figure 9.3: An example test case. A BML block (top) is sent to a RUT and the test function awaits the end feedback for the block (using the `waitForBMLEndFeedback` Test Utility Method). It then verifies the correctness of the execution using various assertions. The Custom Assertions `assertSyncsInOrder`, `assertAllBMLSyncsInBMLOrder`, and `assertBlockStartAndStopFeedbacks` verify the message flow and behavior execution. They validate respectively that feedback on the sync points of the `speech1` and `nod1` behavior was received in the correct order, and that the performance start and stop feedback for the block was received once. The BML block specifies that sync point `speech:start` should occur at relative (to the start of the block) time 6, and that sync point `speech1:syncstart1` should occur at the same time as `nod1:start`. The Custom Assertions `assertRelativeSyncTime` and `assertLinkedSyncs` verify these scheduling constraints. Finally, the Custom Assertions `assertNoExceptions` and `assertNoWarnings` verify that the block was executed without failure.

RealizerTester provides several Test Utility Methods and Custom Assertions [193] to help a test author with this. In the fixture setup phase, the RUT is coupled to a feedback handler that stores all feedback messages. Most of the Test Utility Methods and Custom Assertions act upon these stored feedback messages. The Custom Assertions in RealizerTester verify various commonly required assertions on the received feedback and provide meaningful error messages if these assertions fail.

Figure 9.3 shows an example test case consisting of a BML block (top) and a test function executing the BML block and verifying the assertions implied by the block (bottom). Note that the test case is fully specified using Custom Assertions and Test Utility Methods, that make it very readable.

Many Realizers support custom BML behavior elements. Such elements can be tested using test cases in the Realizer Specific Framework. The Custom Assertions and Test Utility Methods described above can help in the creation of such test cases.

9.4 Testing in Elckerlyc

The BML specification is an emerging standard, and at the moment of writing, there are no Realizers that fully implement the BML/feedback interface proposed by the SAIBA initiative. Elckerlyc implements several (but not all) BML behaviors and supports BML feedback. This makes it a good test-candidate for RealizerTester. Here we describe our experiences with the integration of RealizerTester into Elckerlyc's software development process.

We have implemented an Elckerlyc Specific Testing Framework that sets up a Fixture that uses Elckerlyc as its RUT. Elckerlyc is tested using the 19 test cases provided by RealizerTester. An additional 12 test cases were implemented to test BML behaviors that are specific to Elckerlyc.

Automatic testing has proven useful in both finding errors in Elckerlyc and making sure that new functionality did not introduce errors. In some cases it was useful to define test cases as acceptance tests for new functionality *before* it was implemented.[5] One such test highlighted deficiencies in Elckerlyc's BML scheduling algorithm. Passing the test (by an update to the scheduling algorithm that fixed these deficiencies) marked the implementation of a certain software requirement.

Automatic testing is more valuable if it is done as often as possible. However, running all test cases on RealizerTester takes some time (roughly 3 minutes on our test set of 31 tests), which might discourage its frequent use by Elckerlyc's developers. We have solved this issue by running the tests automatically on Elckerlyc's continuous integration server[6] whenever a developer commits changes to its source repository. If a test fails, the developer responsible for the test failure is automatically notified. The integration server also keeps track of the test performance over all builds, so it is possible to identify exactly which build introduced an error. RealizerTester helps the Elckerlyc developers in the notification of errors, but it does not directly help in identifying the exact location of errors, since it is testing the Realizer as a black box. The use of white box testing at a smaller granularity helps in Elckerlyc's defect localization. To this end, over 1000 unit tests (typically testing one class) and mid-range tests (testing groups of classes working together) are employed to test Elckerlyc. The unit tests run fast (in under 10 seconds) so developers run them very frequently to check the 'health' of newly created code. The test cases by RealizerTester are used in Elckerlyc to test how different (unit tested) components work together as a whole, and, if tests fail, as an indication of locations that require more unit testing.

Automatic testing is useful because it can be done as often as possible without cost (e.g. in developer/human tester time). However, we have found that manual inspection is more flexible than the rigid assertion verification employed by automatic tests and that some errors in Elckerlyc can currently only be identified by manual inspection of the behavior of a virtual human. Therefore we recommend regular visual inspection in addition to automatic testing.

Most visualization failures that have occurred in Elckerlyc so far are a result of

[5] This is a common practice in the Test Driven Development software development process [193].
[6] Elckerlyc uses Jenkins (`http://jenkins-ci.org/`).

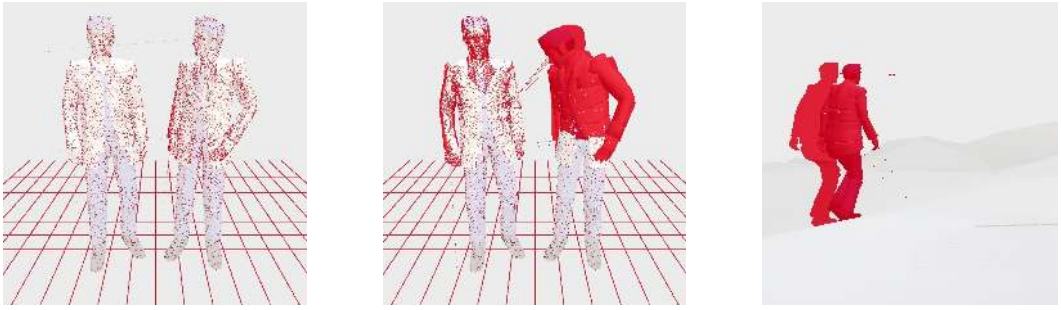
physical simulation errors, resulting in gross movement errors (e.g. the virtual human falling over, rolling through the scene uncontrollably, showing large 'hitches' in movement, etc.). The occurrence of such failures is often dependent on the specific set of movement combinations and the virtual human embodiment used. Because of this, there is often a long time span between the introduction of such failures in the code base and their discovery, which makes the failures hard to repair. SmartBody's testing framework contributes automatic visual regression testing (discussed in the next section). Integrating such automatic visual testing in Elckerlyc would be quite helpful to detect these and other visual regressions in a timely fashion.

9.5 Testing in SmartBody

SmartBody contributes a test program that provides automatic *visual* regression testing mentioned above. This test program takes screen snapshots at predefined moments in an ongoing simulation (e.g. the execution of a BML script). A baseline of such screen snapshots is saved as input for subsequent test simulations. During a later simulation, another snapshot at the same virtual time in the simulation is taken and then compared pixel by pixel against the baseline image. If the images differ more than a predefined threshold (see below), then the test can be marked as failing and examined manually by a tester. To this end, the test program can provide the tester with an image that shows the baseline image overlayed with the differences from the test snapshot. Fig. 9.4 shows some examples of such difference images. A similar visual difference regression testing method is used in the graphics industry [314].

By comparing the results of all aspects of the simulation via a graphical image, the tester is better able to determine the impact of various changes to the realizer. When implementing this method, it is important to position the camera where it can detect meaningful differences between the images during a test run. For example, to test head nodding, the camera should be positioned close to and to the side of the virtual human's face. Also, randomness during simulations, such as reliance on real-time clocks, needs to be eliminated in order to generate repeatable results. Different platforms and graphics drivers will also tend to produce similar, but not identical results. This problem can often be mitigated by setting a sufficiently high image comparison threshold. Changes in functionality will often change the results of the test images that are desirable. If this happens, the tester should then create a new set of baseline test images based on the new functionality. Authoring such a graphical test thus involves choosing a certain simulation, defining a set of important times to check for image differences, defining a camera position and setting an image comparison threshold (using the default value often suffices).

SmartBody's test program goes beyond the tests performed by RealizerTester by checking if the correct motion is generated, rather than by checking if the Realizer sends the correct signals. However, unlike RealizerTester, SmartBody's testing system is Realizer specific and limited to provide only regression (and not acceptance) testing functionality. This testing method is thus complementary to testing



(a) (b) (c)

Figure 9.4: Fig. 9.4a: The simulation run differs from its baseline mostly by subtle differences in the position of the triangular mesh that represents the virtual human's body parts. These differences should not surpass the comparison threshold and should result in a successful test. Fig. 9.4b: The character on the right differs from the baseline in the amount of forward lean towards the gaze target. This difference should exceed the comparison threshold and result in a failed test. Fig. 9.4c: A test of SmartBody's locomotion system on uneven terrain. The baseline is shown on the right, the results from the new test (with a different parameter configuration for e.g. walking velocity) is shown as a silhouette of the virtual human on the left.

by RealizerTester.

Therefore, RealizerTester is also used to test SmartBody. To this end, SmartBody has been extended to allow the feedback messages required by RealizerTester, and our (draft 1.0) test scripts have been converted to the Vienna draft BML standard used in SmartBody. Allowing SmartBody to be tested with RealizerTester involved creating a SmartBody Specific Testing Framework and a SmartBody adapter of RealizerPort (see also Section 9.3.1). This was a relatively simple effort, taking roughly one day of programming. The BML standard only specifies the information that should be contained in the feedback, but does not specify the exact form/syntax of feedback. As a result, much of the implementation effort was spent in parsing SmartBody feedback and converting it to a suitable form for RealizerTester. The process of connecting RealizerTester to a new realizer is similar to connecting any behavior planner to a new realizer. This means that behavior planner developers have to implement error prone and somewhat elaborate feedback parsing for each realizer they connect their behavior planner to. We strongly suggest the incorporation of a standard syntax (for example in XML) for feedback in the BML standard to alleviate this issue. By testing SmartBody with RealizerTester, some minor implementation issues in SmartBody were discovered. We did not find any interpretation differences in the constraint satisfaction between SmartBody's and Elckerlyc's realization of the test scripts. A minor difference in message flow was found. In Elckerlyc, sync-point progress feedback messages are guaranteed to be sent in order (first start, then ready, then stroke_start, then stroke, then stroke_end, then relax, finally end). The performance start messages of a BML block is guaranteed to occur before all sync-point progress feedback messages of the behaviors in the block. The performance stop message of the BML block is guaranteed to occur after all sync-point progress feedback messages are sent. SmartBody does not enforce such

a message order; sync-point progress feedback messages and/or performance start/stop feedback messages that occur at the same time are sent in an undefined order.

9.6 Conclusion and Discussion

We have provided a corpus of BML behaviors and video material of their realization in two Realizers, and a corpus of BML test cases. The test corpus and the visual comparison movie and script corpus are available online under a creative commons license.[7]

Preliminary inspection of the video corpus shows some expressivity issues in the BML standard, but also shows that several behaviors are executed on the two different Realizers in a semantically equivalent manner. More importantly, the process of creating the corpus created healthy competition between different groups building Realizers, each trying to enhance the animation quality of their Realizers to outdo the other. It also motivated them to move toward more compliance to the BML standard.

The modularity proposed by the SAIBA framework not only allows the reuse of existing Realizers with new SAIBA Behavior Planners, but also reuse of test functionality and test cases for different Realizers. The same modularity that allows one to connect a SAIBA Behavior Planner to any Realizer, also allows RealizerTester to test any Realizer. RealizerTester and its test cases provide a starting point for a test suite that can test the BML compliance of Realizers. Such compliance tests are common for software that interprets XML.[8]

When designing BML test scripts *and* their corresponding test assertions we ran into several cases in which the BML specification lacked detail, was unclear, or was unfinished. For example, the current BML specification does not state whether two posture behaviors (using different body parts) could be active at the same time, if gaze can be a persistent behavior, or how custom sync points in a behavior are to be aligned in relation to its default BML sync points. The process of designing a test set of BML scripts and their corresponding test assertions can significantly contribute to the improvement of the BML standard itself by highlighting such issues.

Currently each test case consists of a BML script (in a separate XML file) and a test function in RealizerTester. It would be beneficial to merge the test function itself into the BML script, so that new tests can easily be authored without modifying the source code of RealizerTester itself. For many BML scripts (e.g. those that do not deliberately introduce error feedback or change the behavior flow), it should even be possible to automatically generate test assertions directly from the script rather than authoring them by hand.

The BML standard contains several open issues and its interpretation may vary between Realizer developers. If adapted by multiple Realizers, RealizerTester can contribute to the formalization of the BML standard (by providing BML test scripts

[7] `http://sourceforge.net/projects/realizertester/`

[8] For example, for COLLADA (`http://www.khronos.org/collada/adopters/`) or XHTML (`http://www.w3.org/MarkUp/Test/`) interpreters.

and a formal description of their constraints, as expressed in test assertions) and help identify and resolve execution inconsistencies between Realizers. A Realizer does not need to be fully BML compliant to be tested by RealizerTester; supporting feedback and some BML behaviors is sufficient.

Chapter 10

Elckerlyc in Practice

Elckerlyc features a modular and flexible design, which allows continuous interaction. This chapter illustrates how this design has worked out in practice so far, and, at the same time, illustrates how Elckerlyc's design was partly motivated by the needs of its users. These needs often form the requirements of the several virtual human applications, tools, and user experiments created by them.

In this chapter, I discuss several extensions and tools that make use of Elckerlyc's modularity and flexible input/output connectivity. I also show how Elckerlyc has been employed in several virtual human applications and was used as a research tool in experiments. Providing up-to-date and accurate user documentation is an ongoing challenge. I discuss how Elckerlyc's documentation strategy aims to provide relevant, up-to-date and concise documentation that has a high priority for Elckerlyc's users.

10.1 Modularity

Elckerlyc is designed to be very modular. It allows flexible input handling which provides connectivity through different input channels, and makes it possible to filter or record the BML stream. Elckerlyc can be configured to be used with custom Embodiments and environments. It is also designed to allow one to add custom behavior types and output modalities (through new Engines). This section illustrates these capabilities by showing how they made several extensions of Elckerlyc possible.

10.1.1 Pipes and Adapters

Elckerlyc's BML RealizerPort (see Chapter 8.16.3) allows one to set up and change the 'wiring' between Elckerlyc and a SAIBA Behavior Planner, without modifying the source code of either. Elckerlyc directly implements the RealizerPort and can therefore be used directly (through function calls) by a SAIBA Behavior Planner that uses a RealizerPort to communicate with a Realizer. The SEMAINEToBMLRealizerAdapter and BMLRealizerToSEMAINEAdapter (Figure 10.1) were implemented

to allow connectivity between Elckerlyc and a SAIBA Behavior Planner using the SEMAINE messaging API. The TCPIPToBMLRealizerAdapter and BMLRealizerToTCPIPAdapter allow connectivity between Elckerlyc and a SAIBA Behavior Planner using a custom TCP/IP protocol. When building and deploying applications with Elckerlyc it was often useful to switch to an alternative wiring. A function call connection between the SAIBA Behavior Planner and Realizer is useful during development of an application, when both the Realizer and BehaviorPlanner run on the same computer. In interactive demos however, the SAIBA Behavior Planner and Realizer often run on different computers to ensure that they both run smoothly. In such a setup one of the other connection types (SEMAINE, TCP/IP) is used.

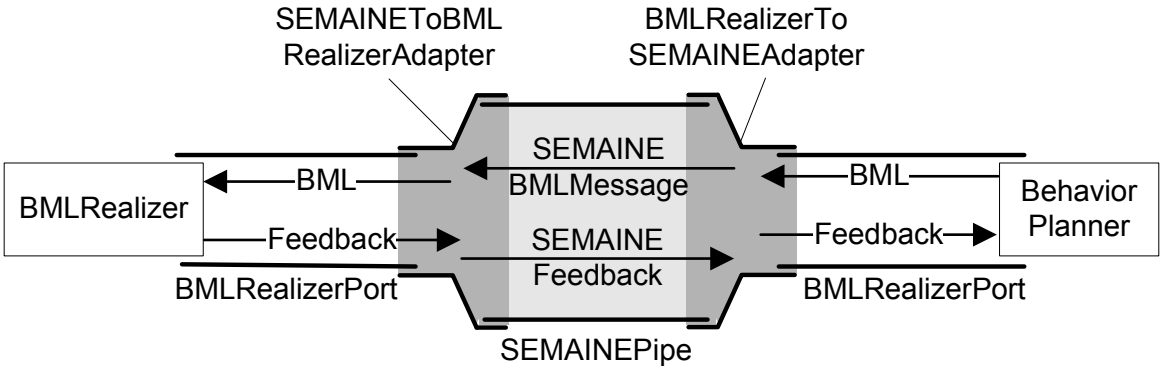


Figure 10.1: The SEMAINE Adapters.

Pipes are used to intercept BML and feedback, allowing one to measure them, let them go through slightly modified, or at a different rate. Pipes were implemented for logging and asynchronous communication with a RealizerPort. The RealizerPort directly connect to Elckerlyc implements the handling of a BML block synchronously, that is, it blocks until the BML block is scheduled. The BlockingQueuePipe can be connected to a RealizerPort to allow asynchronous communication. Scheduling feedback messages (indicating that scheduling has started or is finished on a block, see also Chapter 6.7.1) provide the matching callbacks for such asynchronous communication. The LogPipe (Figure 10.2) is used to log the BML and feedback passing through it. A custom SAIBA Behavior Planner was built to replay these logs. Such playback allows one to carefully analyze recorded user interaction. It can also be used as fake user input during development of a virtual human application, for example, to avoid having to go into a full interaction with the whole system for a quick test of the behavior realization part.

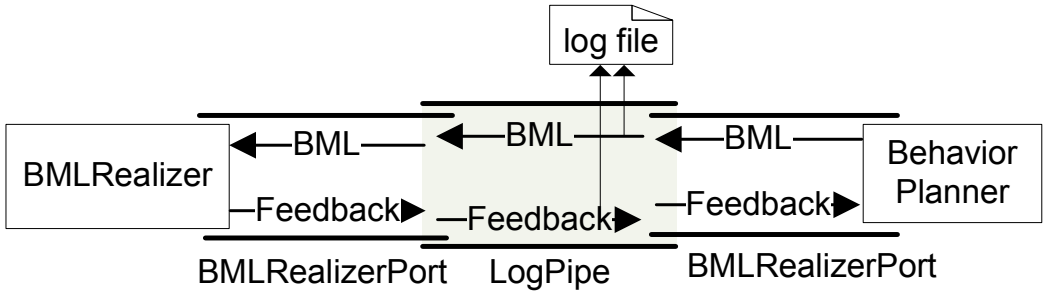


Figure 10.2: The LogPipe.

The Realizer testing framework RealizerTester (see Chapter 9) also makes use of the RealizerPort. RealizerTester is implemented as a SAIBA Behavior Planner that interfaces with a Realizer through its RealizerPort. Allowing RealizerTester to test a Realizer entails creating an adapter that connects this Realizer to the RealizerPort (see Figure 10.3).

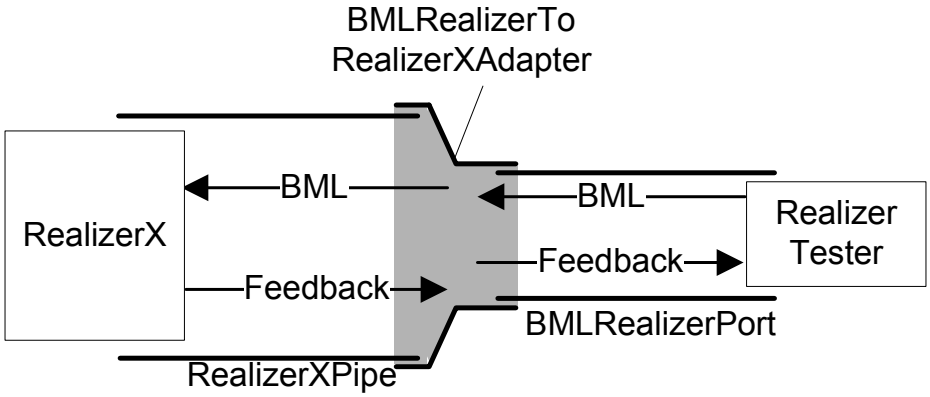


Figure 10.3: Using an adapter to connect RealizerX to the RealizerTester.

10.1.2 Embodiments and Environments

Elckerlyc's Engines set control primitive values on Embodiments, using the specific Embodiment interfaces for the Embodiments they control (e.g. FaceEmbodiment, SkeletonEmbodiment, AudioEmbodiment). New Embodiment implementations can thus be coupled to an Engine that uses the Embodiment interface they implement. Several Embodiments are implemented in Elckerlyc's default setup. Here I discuss ongoing and realized work that connects Elckerlyc to new Embodiments.

Elckerlyc's FaceEngine can steer an Embodiment for the face, using MPEG-4 control primitives. This allows Elckerlyc to steer any face that supports the MPEG-4 standard. An example MPEG-4 Embodiment implementation is provided for one such face: the XFace [15] talking head.

The ThriftSkeletonEmbodiment (see also Chapter 8.16.5) was developed to integrate a SkeletonEmbodiment with *any* desired rendering environment. Ongoing work aims at integrating Elckerlyc with re-lion's Small Unit Immersive Training

(SUIT)[1] environment. SUIT provides an immersive (using a head mounted display, instrumented weapon, sensor suit, and wifi laptop) training environment for law enforcers and soldiers. The need for this integration motivated the design of the ThriftSkeletonEmbodiment. A proof-of-concept implementation of the ThriftSkeletonEmbodiment is currently available for the Ogre rendering environment.[2]

10.1.3 Engines

Elckerlyc allows one to flexibly set up the set of Engines that execute certain BML behaviors. This section discusses how this flexibility was employed in custom Engines for the execution of gesture and facial animation.

10.1.3.1 Elckerlyc on a Mobile Phone: The PictureEngine

Mobile phones do not always have the processing power to run the full 3D environment that is used by default by Elckerlyc. This motivated the design of the PictureEngine (Figure 10.4). Rather than rendering gestures and facial expressions on a 3D body, the PictureEngine uses pictures, or sequences of pictures, to display these behaviors.

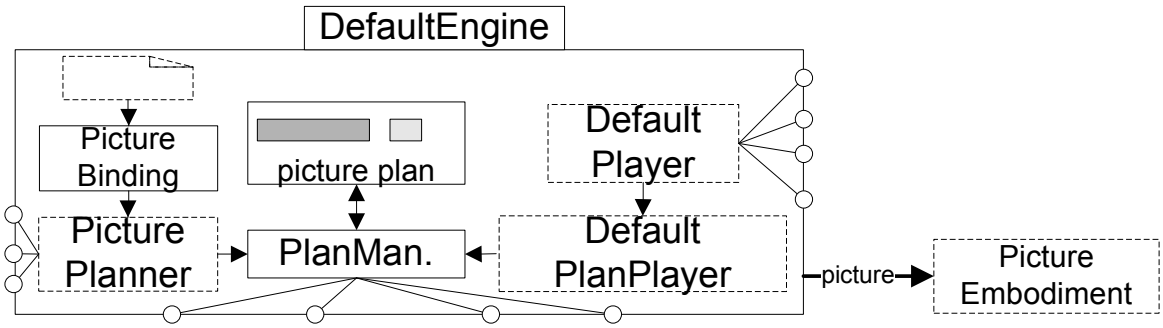


Figure 10.4: The PictureEngine.

10.1.3.2 Nabaztag Integration: the Nabaztag Engine

The Nabaztag is a robot rabbit with ears that are controlled by servomotors and a body on which colored led lights are displayed. The SERA demo (see Section 10.3.5) required control of this rabbit using BML, without encumbering Elckerlyc itself with Nabaztag specific code and libraries. The Nabaztag Engine (Figure 10.5) is designed to achieve this. It is registered for handling all non-speech behaviors. For example, head nods are mapped in the Nabaztag Engine to a NabaztagPlanUnit that would move the ears shortly forward and back again; a sad face expression is mapped to a NabaztagPlanUnit that let the ears droop; et cetera.

[1] http://www.re-lion.com/products/suit
[2] http://www.ogre3d.org/

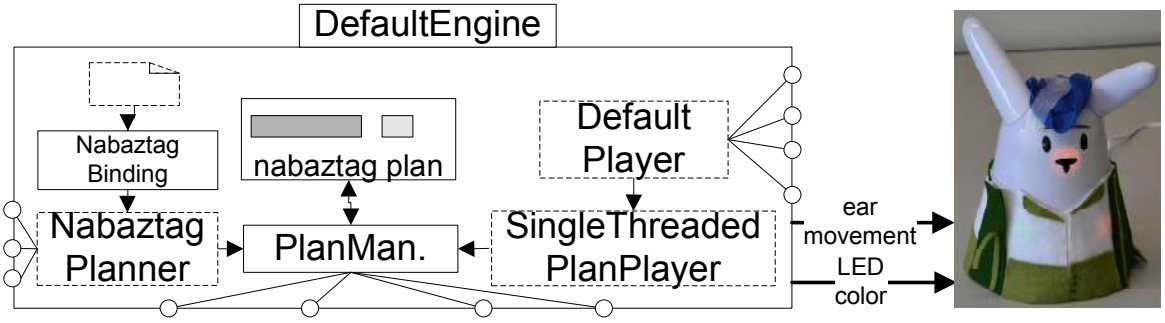


Figure 10.5: The Nabaztag rabbit and the Engine steering it.

The control primitives for the Nabaztag robot are (1) move the ears of the robot to a specified position, (2) move the ears forward or backward by a specified amount, and (3) set one of the LEDs to a certain color. So far, two PlanUnit types have been implemented. The 'MoveEarTo' PlanUnit moves the ears of the robot to a specified position by linear interpolation during the duration of the PlanUnit. The "WiggleEarTo" PlanUnit interpolates the ear from its current position to the specified target position and back to the starting point, during the duration of the PlanUnit, using a sinusoid interpolation. Given these PlanUnits, and a NabaztagBinding for mapping BML behaviors to Nabaztag PlanUnits, the Nabaztag Engine is constructed using the standard available Engine components (see Figure 10.5).

The Nabaztag extension was achieved in a matter of days and did not require any changes in Elckerlyc's source code.

10.2 Asset Creation

Elckerlyc comes with a virtual character and a set of body and facial animations. New applications typically require the use of their own virtual character and a set of application specific animations. Here I discuss what needs to be done to make such a virtual character compliant with Elckerlyc, and how new animations can be authored using custom tools or commercial animation software.

10.2.1 Hooking up Elckerlyc to a New Virtual Human Mesh

Often, virtual applications require a custom-built virtual human body. Elckerlyc's default render environment can import meshes in the royalty free XML format COLLADA.[3] COLLADA is the industry standard format designed to allow exchange of assets between different applications in the 3D industry and is thus widely supported by different vendors. For performance reasons (e.g. loading time), a COLLADA mesh can be exported to a custom binary format.

If one wants to make use of physically simulated animation, it is necessary to design a model of the physical body of the virtual human. This model describes the set of rigid bodies that compose the physical body and the joints that connect these

[3] `https://collada.org/`

rigid bodies. Each rigid body is described by its mass, inertia tensor, center of mass and collision shape. Each joint is described by its degrees of freedom and the joint limits on these degrees of freedom. Elckerlyc's physical human editor (Figure 10.6) provides the functionality to author a physical humanoid. Convenient defaults for the joint rotation limits of males and females (based on the human factors literature [310]) are provided for each joint. A simple (and thus fast to simulate) collision shape or combination of shapes can automatically be fitted to the (sub)mesh corresponding with each rigid body. The center of mass, inertia tensor and mass of rigid bodies are automatically set (using [195]) by importing the custom mesh of the rigid body and setting its density. The physical human editor provides default densities for each body part based on [307] and [67]. The custom mesh of each rigid body has to be constructed manually from the original mesh of the virtual human. This is done by segmenting the mesh of the original virtual human, adapting it to be skintight, and closing the gaps in the resulting segments.

In order to have full control of the virtual human's face expressions, one needs to make the face MPEG-4 compliant using Elckerlyc's face editor (see Figure 10.7). For each of the standard MPEG-4 control points of a face, one needs to define their effect on the mesh.

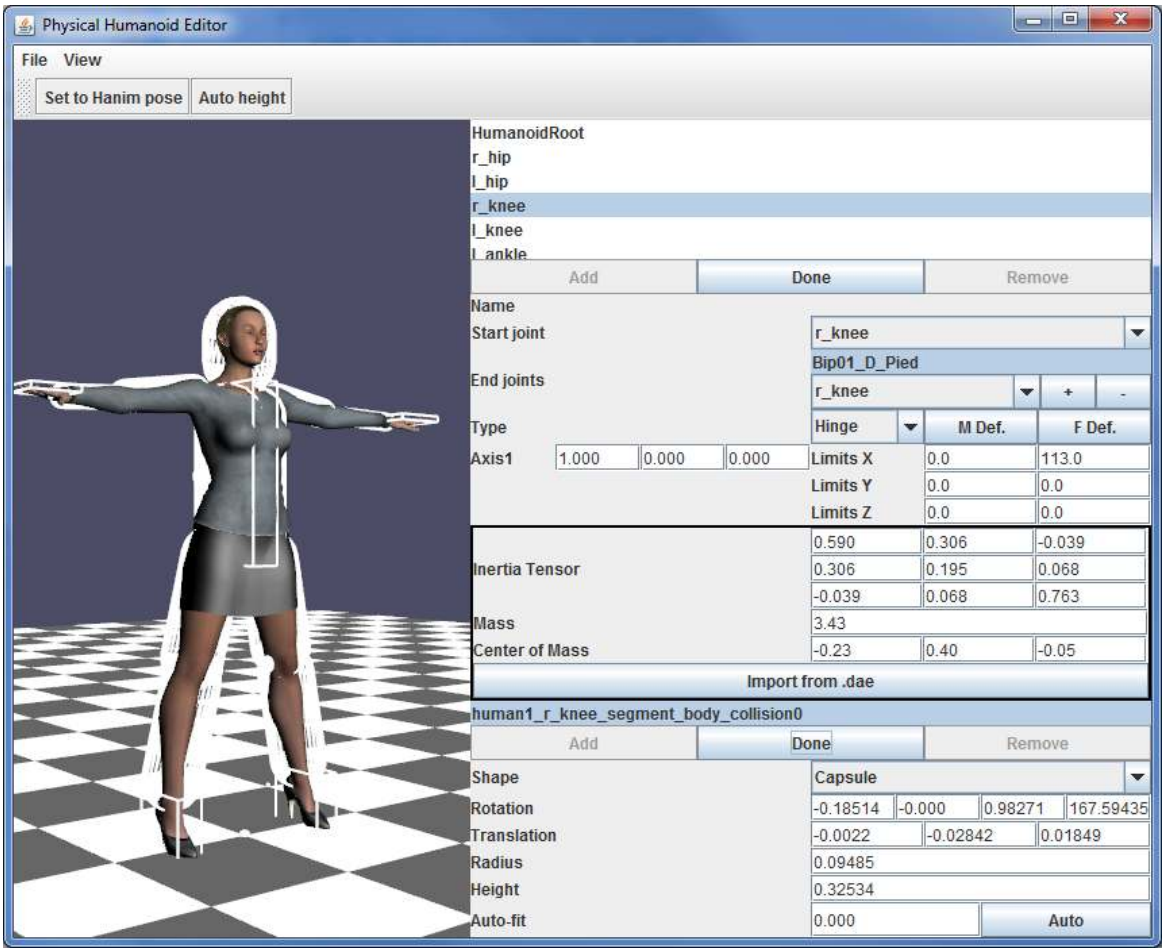


Figure 10.6: The physical human editor.

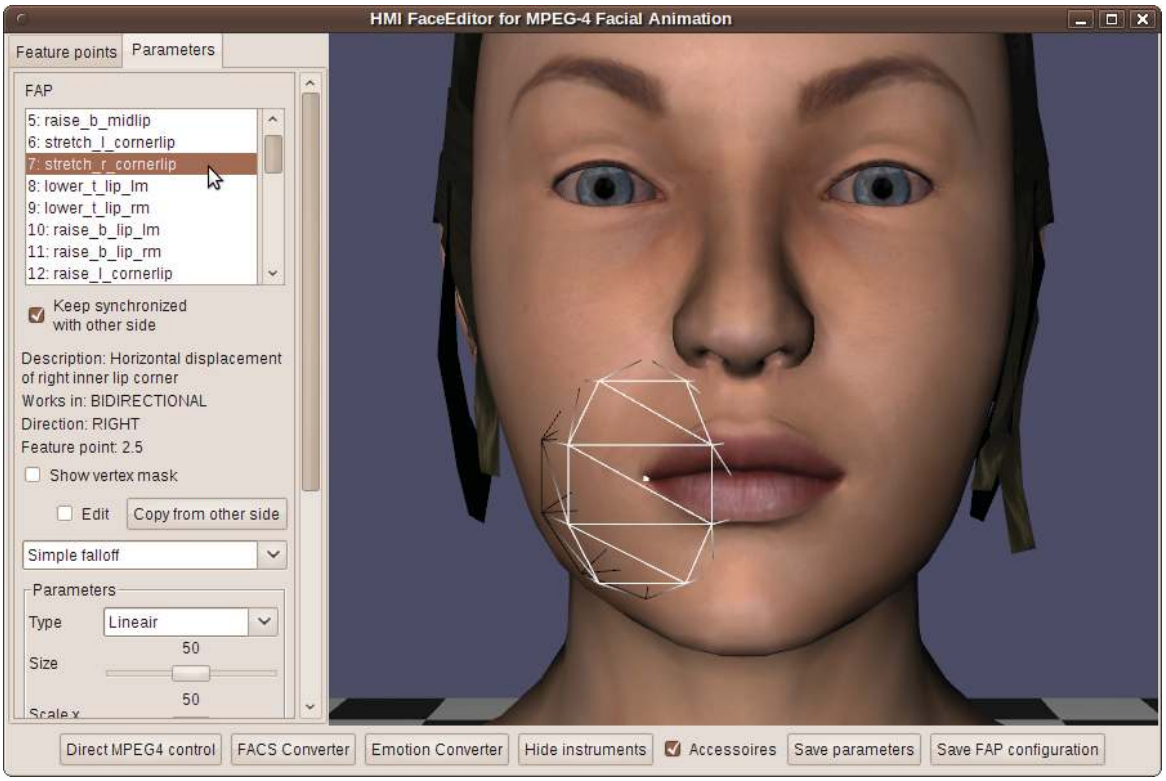


Figure 10.7: The MPEG-4 face editor.

10.2.2 Authoring Animation

10.2.2.1 Motion Capture and Keyframe Animation

Elckerlyc provides an XML format to store keyframe animation. Animations (or motion capture) created with Autodesk MotionBuilder[4] can be converted to this XML format using a custom plugin. Simple animations (for e.g. hand poses, see Figure 10.8) can be written manually in a relatively easy manner.

10.2.2.2 Creating Procedural Animation

Elckerlyc provides a specialized editor to author procedural animation (see Figure 10.10). It provides functionality to specify the key times of an animation, add animation parameters, import motion capture animation, author the movement path of end effectors (e.g. the wrist, feet) and author the rotation path of joints. These movement and rotation paths are authored using mathematical formulas of time and the specified parameters.

Simple procedural animations (e.g. those dealing with a small number of joints/ end effectors) have also been created by manually editing their XML files. Figure 10.9 shows an example of a procedural animation that was created in this manner.

[4] `http://usa.autodesk.com/`

```
<SkeletonInterpolator rotationEncoding="axisangles" encoding="R" parts="r_thumb1
  r_thumb2 r_thumb3 r_pinky1 r_index1 r_middle1 r_ring1 r_pinky2 r_middle2
  r_index2 r_ring2">
0 1 0 0 0.6      1 0 0 0.6     1 0 0 0.4
  0 0 1 1.57     0 0 1 1.57    0 0 1 1.57    0 0 1 1.57
  0 0 1 1.57     0 0 1 1.57    0 0 1 1.57    0 0 1 1.57
1 1 0 0 0.6      1 0 0 0.6     1 0 0 0.4
  0 0 1 1.57     0 0 1 1.57    0 0 1 1.57    0 0 1 1.57
  0 0 1 1.57     0 0 1 1.57    0 0 1 1.57    0 0 1 1.57
</SkeletonInterpolator>
```

Figure 10.8: A keyframe animation of a static clenched fist. Contains two keyframes, (at time 0 and 1). Each joint rotation is described by a rotation axis and a rotation angle.

```
<ProcAnimation prefDuration="1.0" minDuration="0.8" maxDuration="1.5">
  <Rotation rotation="if(t < 0.1,0,a * sin((t-0.1) / 0.9 * pi * r));
    a * 0.5 * sin( t * pi * r);
    if (t < 0.1,0, 0.25 * -a * 0.5 * sin((t-0.1) / 0.9 * pi * r))"
  target="vc4"/>
  <Rotation rotation="if(t < 0.1,0,a * sin((t-0.1) / 0.9 * pi * r));
    a * 0.5 * sin( t * pi * r);
    if (t < 0.1,0, 0.25 * -a * 0.5 * sin((t-0.1) / 0.9 * pi * r))"
  target="skullbase"/>
  <Parameter sid="a" description="magnitude" value="0.25"/>
  <Parameter sid="r" description="repeats" value="1"/>
  <KeyPosition id="ready" time="0.3" weight="1.0"/>
  <KeyPosition id="relax" time="0.7" weight="1.0"/>
  <KeyPosition id="stroke" time="0.5" weight="1.0"/>
</ProcAnimation>
```

Figure 10.9: A hand authored procedural animation: a head nod that is inclined to the left.

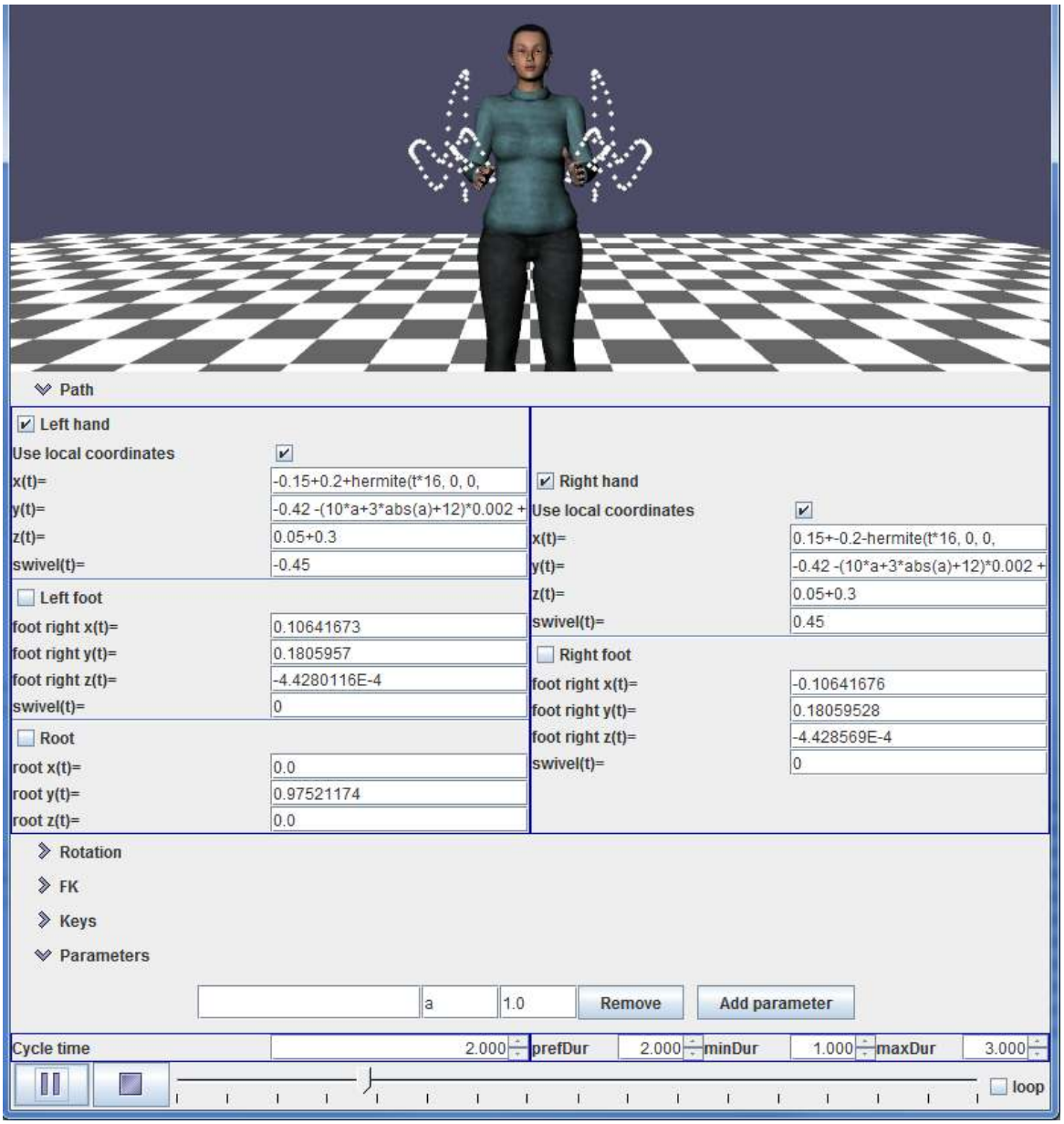


Figure 10.10: The procedural animation editor.

10.3 Applications

Elckerlyc has been used in several applications with different interaction requirements, ranging from no interaction (video playback) to continuous interaction. In this section, I provide an overview and discuss how these applications have influenced and motivated Elckerlyc's design process.

10.3.1 DirectLife Integration

Philips' DirectLife program[5] aims at increasing the daily activity level of its users. It keeps track of the physical activity of a user wearing an activity sensor. It helps in setting activity goals and tracks progress. In a current prototype, Elckerlyc is connected to DirectLife to provide users with information on their activity progress and suggestions to become more active. Here Elckerlyc is used in a non-interactive setting: information on activity progress is translated into BML and shown to users in a video (see Figure 10.11).

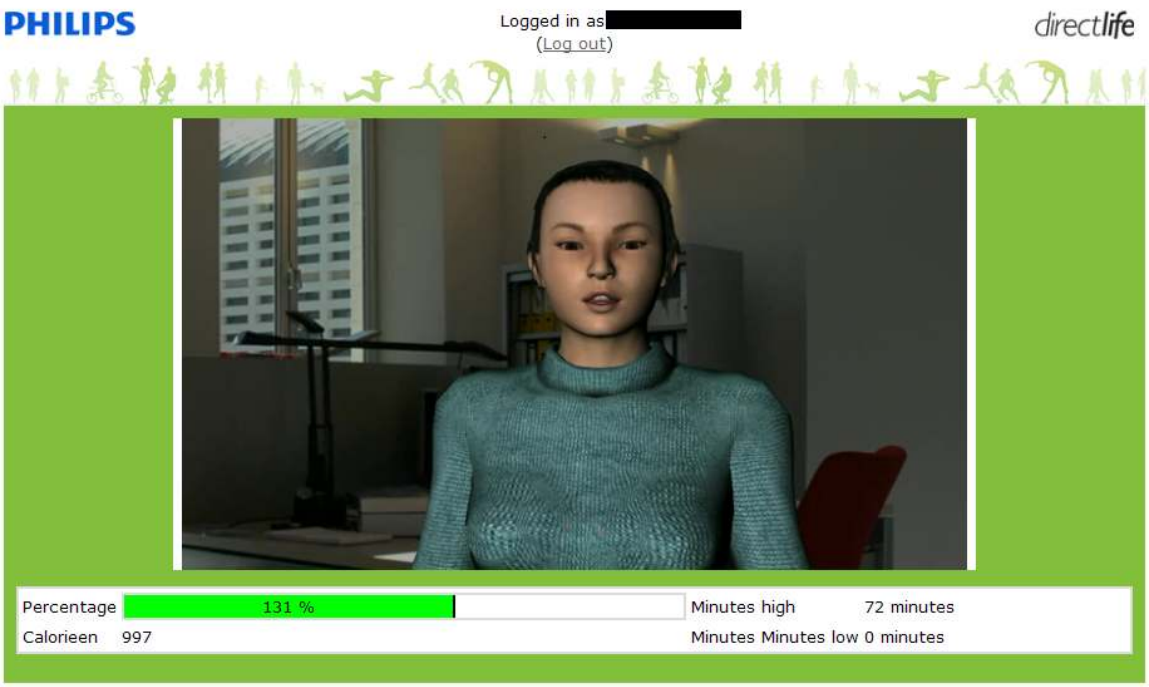


Figure 10.11: Elckerlyc used in DirectLife.

10.3.2 SimQuest Integration

SimQuest [127] is an inquiry based learning environment, in which students actively discover information by allowing them to experimenting freely with parameters in a computer simulation. For example, students can change the mass of two

[5] http://www.directlife.philips.com/

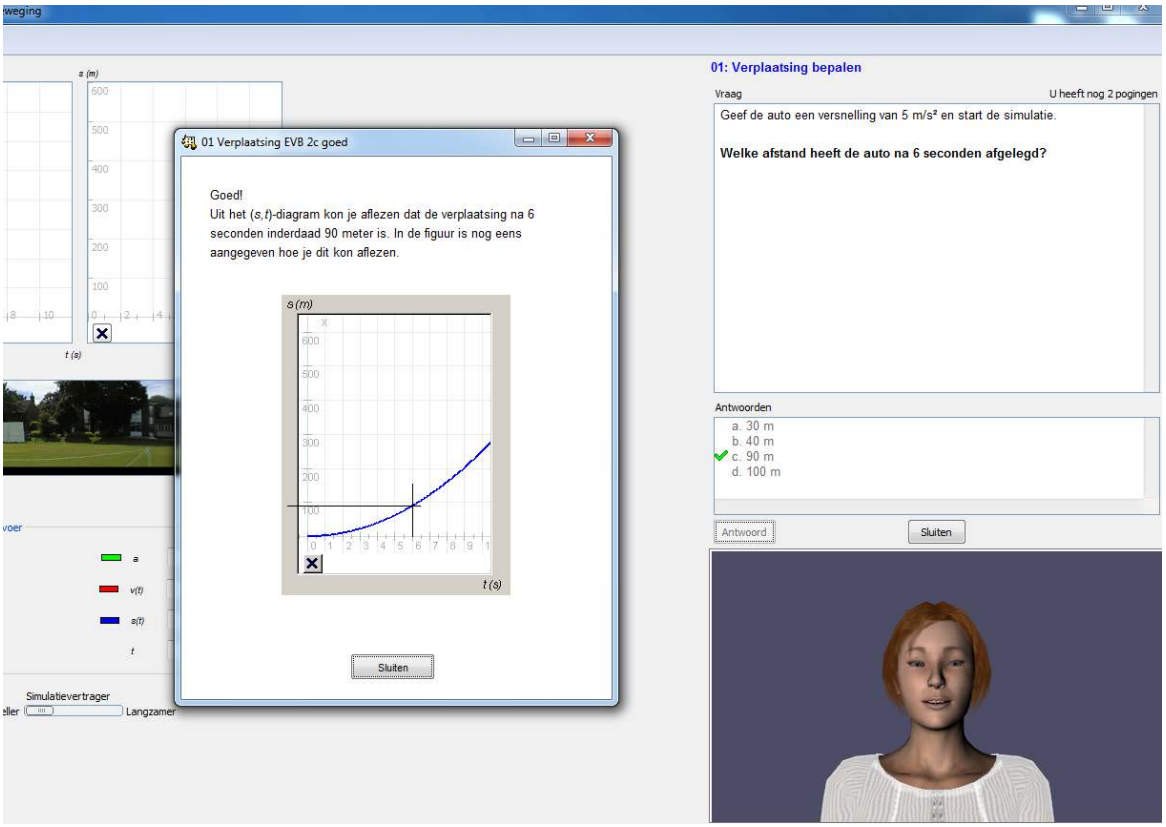


Figure 10.12: Elckerlyc used in SimQuest.

people and their distance from the center of a seesaw and discover the effect on the momentum.

Elckerlyc is integrated with this environment as a talking head (see Figure 10.12). Currently it is used to explain the ongoing experiment or to give hints or other feedback on it. Later work could employ Elckerlyc's continuous interaction capabilities to provide feedback that is tightly coupled with an ongoing simulation and/or user actions.

10.3.3 Griet: the Girl with a Pearl Earring

Griet: the girl with a pearl earring (Figure 10.13), is an interactive demo using Elckerlyc, demonstrated at the New Technology Conference in Amsterdam. Griet, painted by Johannes Vermeer as the girl with a pearl earring (1665), tells about her life as a maid in Vermeer's house in Delft. Moreover, visitors can ask her questions about her life in the 17th century. Griet features gazing and gestural behavior: Griet follows you with her eyes, and makes gestures at appropriate times. By embodying historic people, and by providing interaction with them, this demo aims to more actively engage the audience in the history of these people. Users can interact with Griet through selections in a multiple choice menu, which is updated after each 'turn' in the conversation (this is similar to the conversation mechanism employed in [30]).

Figure 10.13: Interaction with Griet

10.3.4 Psychometer

The psychometer [34] is a validated instrument for the measurements of personality traits of a human being. To determine the personality of the user, the psychometer asks a set of questions to which the user can answer on a scale of 1 to 5. The psychometer was later extended to allow question answering using natural language [289]. This version of the psychometer allows users to ask for clarifications of the question and asks follow-up questions if the answer from the user cannot readily be classified in one of the 5 categories it needs. The latest version of the psychometer [275] is implemented as a talking head using Elckerlyc. This talking head makes use of gaze behavior (e.g. gaze away to mark the start of the sentence and gaze up to mark 'thinking') and speech synthesis. The psychometer uses a simple form of behavior interruption. While the user is typing an answer, the talking head executes some idle behavior. This idle behavior is interrupted as soon as the sentence is typed in completely, after which the talking head responds with a clarification, a follow-up question or a new question.

10.3.5 SERA Demo

Deliverable 3.3 of the Social Engagement with Robots and Agents (SERA) project [5] provides a demo that integrates Elckerlyc with a dialog system that was designed using a combination of Hierarchical State Machines and the Information State component Flipper [180]. Output is generated through the Nabaztag rabbit (see also Section 10.1.3.2). Users can interact with the demo using user interface buttons with a predefined function (e.g. yes, no, don't know, repeat, change topic. See also Figure 10.14). Input from the user may change the dialog state and interrupt ongoing utterances. The dialog manager needs to keep track of the realization status of the behavior it has requested from Elckerlyc (using Elckerlyc's feedback messages), both to keep track of what information was delivered to the user and to interrupt ongoing behavior in an elegant manner. This demo motivated the Nabaztag integration discussed in Section 10.1.3.2. It also illustrates how adding interruption capabilities to a SAIBA Behavior Planner (in this case a dialog manager) requires it

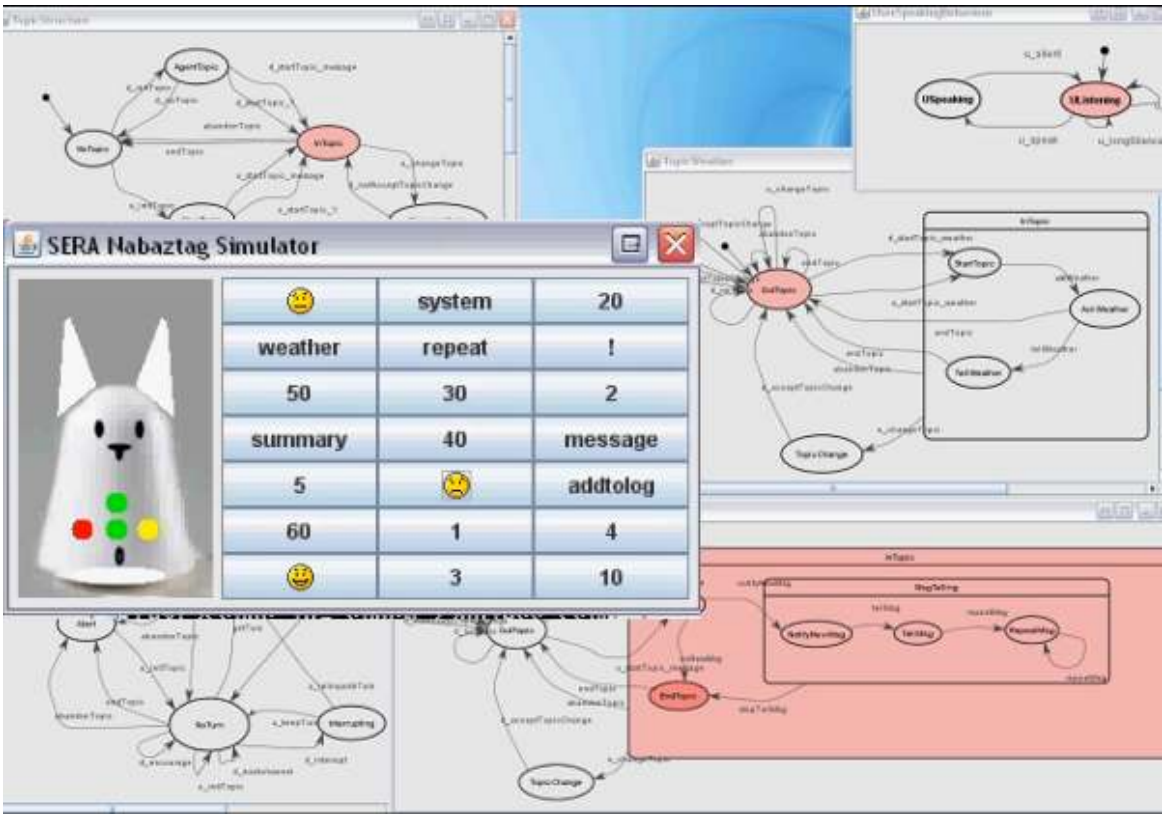


Figure 10.14: User interface and visualization of the dialog state in the SERA Demo.

to keep track of the realization status of the multimodal plan it composed.

10.3.6 Reactive Virtual Trainer

The scenario of the Reactive Virtual Trainer (RVT) describes a virtual human capable of presenting physical exercises that are to be performed by a human, monitoring the user and providing feedback [247, 248]. The reactivity of the RVT is manifested in natural language comments, readjusting the tempo, pointing out mistakes or rescheduling the exercises. Such exercises can be performed at the beat of a user's favorite music. Exercises describe a mix of behaviors on different modalities, including exercise movement, sound (such as clapping, feet tapping), speech and music. This scenario is similar in certain ways to the virtual conductor described in Chapter 8.1.2. The RVT can do the exercises along with the user, adapting its tempo to the performance of the user, or attempting to lead the user when his/her tempo is lagging.

The RVT Trainer has been one of the first applications informing Elckerlyc's design requirements for continuous interaction. Several trainer prototypes have been designed and implemented [247, 248]. The latest version of the trainer [66] can explain exercises and execute them in coordination with a human. The trainer makes use of Elckerlyc's Anticipator specification and a custom ExcerciseAnticipator that predicts key times of user movement in his/her exercise. This ExcerciseAnticipa-

tor makes use of a Nintendo WiiMote sensor.[6] This sensor provides acceleration measurements. Acceleration peaks were found to coincide with key times of user exercises. The ExcerciseAnticipator predicts upcoming exercise key times, using the history of detected acceleration peaks (see [66] for implementation details). These time predictions are typically aligned with key time moments in the exercise of the RVT. The exact alignment strategy is determined by the intent of the trainer (for example, the RVT could move slightly ahead of the user, to make him move faster).

10.3.7 The Attentive Speaker

In the eNTERFACE project on continuous interaction, a first design and implementation for an attentive speaker that uses Elckerlyc was made [228]. The attentive speaker is a virtual human that is able to attend to its interaction partner while it is speaking and modifies its communicative behavior on the fly based on what it observes in the behavior of its partner. Continuous interaction is one of the fundaments underlying attentive speaking.

An attentive speaker has several turn-taking and listener response management capabilities. Depending on the response from a listener, the attentive speaker might fluently interrupt her ongoing speech, attempt to keep the turn (for example by increasing the speech volume), delay her speech for a bit to allow the listener to finish an assessment, or continue speaking before a listener finishes a continuer.[7] These attentive speaking capabilities have motivated the BML^T specifications for pre-planning, interruption and parameter value changes. To guide these attentive speaking capabilities, a rapid interpretation of listener behavior is important. To this end, the eNTERFACE project has contributed the predictive (defined as within a minimal perceptible speech pause) classification of incoming listener speech in Listener Responses and non-Listener Responses.

10.4 Elckerlyc in User Experiments

Elckerlyc was used as a research tool in a series of experiments on listener responses at HMI [148, 223, 224]. All these experiments involved evaluations by test subjects of movies of a (fake) interaction between a human speaker and a virtual listener steered by Elckerlyc. The listening behavior of the virtual listener was either based on annotated behavior of a real listener that interacted with the speaker, on the basis of a listener response model, or set up with some random parameters. It involved head nods, eye blinks and vocalizations (e.g. uh-huh). Since generating vocalized listener responses using text-to-speech systems is still an open research area [214], these experiments used recorded listener responses that can be performed with a closed mouth instead. This need for playback of recorded audio motivated the

[6] `http://www.nintendo.com/wii`

[7] see Chapter 5.1 for more detail on speaker / listener interaction in turn taking and listener responses

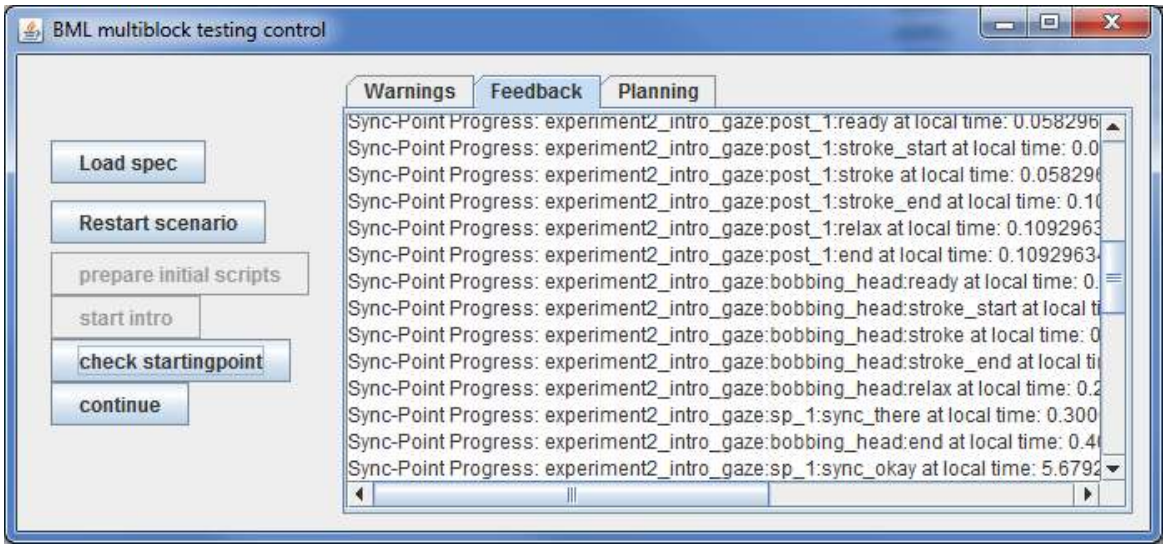
design of Elckerlyc's AudioEngine and the corresponding BML^T audio behavior (see Chapter 8.11).

Other experiments were performed to inform or test the design of some of the applications discussed in Section 10.3. User experiments with the virtual trainer show that it can successfully speed up the movement of users by moving slightly faster than them (see also the scenario in Chapter 6.5.1) [66]. Two experiments were performed with Elckerlyc in the context of the Attentive Speaker application. The experiments and their results are discussed in detail in [228], here I give a brief overview of the Elckerlyc functionality used in them. The first experiment evaluated the effectiveness of different listener responses elicitation strategies. To give subjects the impression that the virtual human could understand their responses, the virtual human asked explicitly for a response at the start of the experiment (e.g. "are you ready?") and appeared to wait for a response (using a Wizard of Oz setup). Other than the explicit acknowledgement request at the start of the experiment, the virtual human did not react to the behavior of the subject. In this experiment, the illusion of Attentive Speaking was broken whenever the Attentive Speaker resumed its speech while the subject was uttering a longer response. The second experiment was set up to allow the Attentive Speaker to wait for longer listener responses. This was done by aligning the start of each utterance of the Attentive Speaker to the release event of a Spacebar Anticipator. The wizard would press the spacebar while the participant was speaking and release it when the participant stopped. The experiments with the Attentive Speaker resulted in several generic tools that can be used for user experiments with Elckerlyc, including a framework for the design of Wizard of Oz tests, the LogPipe and the Spacebar Anticipator. The first two are discussed below, the latter in Chapter 8.5.1.

10.4.1 The Wizard of Oz Testing Framework.

A testing framework was designed to allow the simple authoring of Wizard of Oz user experiments in Elckerlyc. Such experiments are designed using an XML description (Figure 10.15, bottom). This XML description binds a user interface button to a list of BML scripts (see Figure 10.15, top for an example of the resulting user interface). Whenever a button is pressed by the wizard, the associated BML scripts are sent to Elckerlyc. The user interface also allows the wizard to inspect Elckerlyc's execution, planning and warning feedback. Using this simple test setup in combination with Elckerlyc's continuous interaction capabilities, very powerful interaction scenarios can be designed. Buttons can be bound to scripts that pre-plan BML blocks, activate preplanned BML blocks, interrupt ongoing behavior or change parameter values in ongoing behavior.

The example script in Figure 10.15 is a test scenario for the Attentive Speaker. It uses the "prepare initial scripts" button to pre-plan some BML blocks. The other three buttons are used to instantly activate the pre-planned behavior. The "start intro" starts the experiment with an explanation of the experiment by the virtual human. At the end of this explanation, the virtual human asks the test subject if he/she is ready for the experiment. After a confirmation by the test subject, the




```
<elckerlycmultiblocktester>
  <scriptlist name="prepare initial scripts">
    <script filename="experiment2_intro_gaze.xml"/>
    <script filename="experiment2_wait_gaze_a_nowait.xml"/>
    <script filename="experiment2_route1_startingpoint_gaze.xml"/>
    <script filename="experiment2_wait_gaze_b_nowait.xml"/>
    <script filename="experiment2_route1_combined_2_outside_left.xml"/>
    ...
    <script filename="experiment2_route1_combined_9_street_right.xml"/>
    <script filename="experiment2_wait_gaze_9.xml"/>
    <script filename="experiment2_conclusion_gaze.xml"/>
  </scriptlist>
  <scriptlist name="start intro">
    <script filename="fire_experiment2_intro_gaze.xml"/>
  </scriptlist>
    <scriptlist name="check startingpoint">
    <script filename="fire_experiment2_route1_startingpoint_gaze.xml"/>
  </scriptlist>
  <scriptlist name="continue">
    <script filename="fire_experiment2_route1_combined_with_pause.xml"/>
  </scriptlist>
</elckerlycmultiblocktester>
```

Figure 10.15: The Wizard of Oz interface and the XML configuration that created it.

wizard presses the “check starting point” button, which starts an explanation of the starting point. This explanation ends with “Do you remember?”. After a confirmation by the test subject, the wizard presses the “continue” button, which starts the final part of the experiment: an explanation of the route.

10.4.2 Experiment Logging and Replay

When running experiments with an interactive virtual human (interactivity might be achieved using a Wizard of Oz setup), the behavior of the virtual human is not fully predictable beforehand. It is therefore useful to keep track of exact behavior executed by the virtual human for further analysis and/or replay. The LogPipe was designed specifically for this purpose. It is placed between the BMLRealizerPorts of the SAIBA Behavior Planner and Elckerlyc and captures all communication between them (BML and feedback). A specialized SAIBA Behavior Planner was designed to replay the recorded execution logs.

10.5 Documentation

Outdated and/or lacking documentation plagues most, if not all, other Realizers. While their source code is continuously updated, the documentation of the open source Realizers Greta [104], SmartBody [280] and EMBR [111] has not been updated for at least a year. The manual of the closed source Realizer MARC [64] is listed as ‘in development’ and currently only provides a one page installation document.[8] Other software engineering projects typically also fail to achieve high quality, up to date and complete documentation [26, 40]. Some software development processes have tried to solve this documentation problem.

Extreme Programming [25] is a lightweight software development process that proposes to omit software documentation. The software is instead documented using:

1. *Test cases*: acceptance tests capture the requirements specification, unit tests can explain the interface of classes.

2. *Clearly written source code*

3. *Face to face communication*. Programmers work in changing pairs, thus spreading knowledge through the organization. In addition to that, the software development team is small and works in one room, so it is easy to ask questions about the software of a coworker.

Elckerlyc’s documentation is in part provided in source code examples. Software development processes that use test driven development (including Extreme Programming) propose to use test cases as source code examples [193]. Elckerlyc

[8] Documentation as provided on the respective web sites of these Realizers, visited at May 19, 2011.

uses a slightly different setup in which test cases automatically execute and test all provided examples. This allows one to easily reuse and execute the examples without being dependent on the testing framework used. The GUI testing framework fest[9] is used to run example programs, execute some user interface actions on them and verify the results. A visual testing tool such as that used by SmartBody (see Chapter 9.5) could be developed for automatic visual verification of the examples. The examples and their test cases are compiled and tested on Elckerlyc's continuous integration server (see also Chapter 9.4). This ensures an early notification of compilation errors or test failures of the examples. Such errors indicate that the example is no longer up-to-date with Elckerlyc's source code and should either be fixed or removed. By making sure that all examples keep on passing their tests and removing examples that are no longer relevant (typically after they fail to compile or they fail a test), the example set remains up to date and concise.

Virtual human applications using Elckerlyc were so far mostly written by students. While they did not share a room with Elckerlyc developers, or pair programmed with them, Elckerlyc's working was often explained to them most easily in a face-to-face manner. Unfortunately, Elckerlyc developers (like most researchers) are not always available to answer questions on their software. Furthermore, face-to-face communication will become harder once Elckerlyc is used within other research groups. Face-to-face communication does not suffice as a documentation strategy for Elckerlyc; some written documentation should also be provided.

Berglund and Priestly [26] provide some insights on how to set up written documentation in such a way that relevant information of high priority is available for users. They argue that software developers will have to accept that documentation is inherently incomplete. They propose that the focus of documentation should be on providing customers with answers to their questions rather than with providing complete documentation of their software. To this end, they propose a documentation process that is similar to open source software development. Open source software development is characterized (amongst other things) as a development process that is 'user driven' and 'just in time'. Or, in other words, what gets developed is what the users want when they want it badly enough. These two aspects capture both relevance and priority. In a 'user driven' documentation approach, it is essential that users are allowed to contribute to the documentation. Such contributions can come in the form of manuals, questions on a discussion forum or mailing list, through annotation of (online) manuals, and so on. Rather than writing documentation, the task of technical writing staff becomes one of organizing and editing user-provided documentation. The writing staff should also provide a first prototype of the documentation.

Several users of Elckerlyc were willing to contribute pieces of documentation, ranging from one page 'to achieve X I sent this BML' descriptions to a fully fledged beginners' guide. Berglund and Priestly's documentation process thus seems feasible for Elckerlyc. It is employed on Elckerlyc's web site.[10] The first prototype of Elckerlyc's online manual was set up using documentation from the developers,

[9] `http://code.google.com/p/fest/`

[10] `http://elckerlyc.ewi.utwente.nl/`

augmented with user documentation. Elckerlyc's manual supports user annotations. The forums are used for user support, and the answers to user questions and user annotations of the manual are to be integrated regularly with the manual.

10.6 Conclusion

Preliminary versions of Elckerlyc were used by several experimenters and virtual human application designers. These users of Elckerlyc appreciated its flexible design and continuous interaction capabilities. Elckerlyc's users are most critical over the documentation that comes with Elckerlyc. The publicly released version of Elckerlyc improves upon the documentation in the preliminary versions by using a combination of automatic testing of source examples and a process of 'user driven', 'just in time' documentation creation. This should provide relevant, up-to-date and concise documentation that has a high priority for Elckerlyc's users.

The flexibility and modularity of Elckerlyc's design is illustrated by the several extensions that employ it. Elckerlyc was used in applications with varying interactivity needs. Some of these applications demonstrate Elckerlyc's continuous interaction capabilities, and, at the same time, often motivated new requirements or minor improvements in Elckerlyc's functionality. For example, the design of several applications requiring different forms of continuous interaction (the Reactive Virtual Trainer, the Attentive Listener, the Interactive Virtual Conductor, and the Interactive Virtual Dancer) have contributed in shaping Elckerlyc's continuous interaction requirements and design. Elckerlyc has also been used as a research tool in several experiments. The flexibility of its input 'wiring' allowed the implementation of valuable generic user testing tools such as the Wizard of Oz testing framework and the experiment logging and replay tool.

Chapter 11

Conclusion

I have defined three research goals that motivated Elckerlyc's modular design, its continuous interaction capabilities and its adjustable trade-off between motion naturalness and control. Here I discuss the contributions made by this thesis on these topics.

11.1 Enabling Collaboration and Competition in Virtual Human Design

Designing a virtual human is a massive software engineering task that can profit from modular design, in which interfaces are shared by different research groups. The SAIBA framework provides a first step towards shared interfaces for 'standard' components in virtual human architectures. Standardization of the interface of these components allows their reuse by different research groups and allows easy comparison (and thus competition) between research groups that create the same types of components. This thesis contributes to both the definition and formalization of one of the SAIBA interfaces (BML) and provides an implementation of the component that implements it: the Behavior Realizer Elckerlyc. Elckerlyc's modular design enables collaboration opportunities beyond those offered by implementing the SAIBA interface. This design makes it possible for other research groups to easily connect Elckerlyc to their SAIBA Behavior Planner, to add specific modularities (e.g. to control a robot) or to connect it to their own rendering environment or virtual human. Below I discuss the contributions to the BML standard and Elckerlyc's modular design in more detail.

I contributed the first formal definition of the explicit, implicit and cluster scheduling constraints implied by a BML block. I aimed to promote and test the SAIBA compliance of BML Realizers. Work towards this goal was done in a joint effort with the SmartBody authors. We contributed a corpus of example BML scripts and videos of their execution by different Realizers. The video corpus demonstrates the semantic equivalences and differences in execution of BML scripts by Elckerlyc and SmartBody. I also contributed a tool called RealizerTester that can be used to

formally test and maintain the adherence of any Realizer to the BML standard. RealizerTester can 1) help in maintaining the stability and extensibility that is crucial for Realizers, and 2) contribute to the formalization of the emerging BML standard, both by providing test scripts and a formal description of their constraints and by identifying and resolving execution inconsistencies between Realizers.

Elckerlyc was designed to offer modularity beyond that offered by implementing the SAIBA Realizer interface. Elckerlyc provides flexible 'wiring' with a SAIBA Behavior Planner. This wiring allows one to easily change the connection type (e.g. function calls, TCP/IP) and to filter or record the BML stream. Elckerlyc allows one to easily configure and change the Embodiments its Engines steer by alternative Embodiments that implement the same interface. For example, Elckerlyc's FaceEngine can steer any Embodiment that implements the MPEG4Embodiment interface. New Engines can be added to steer new Embodiments (for example a Nabaztag robot rabbit). Bindings, specified in XML, allow one to easily configure the mapping from BML behaviors to units on the plan of a specific Engine. For example, the GestureBinding can be used to easily hook up a custom animation to a specific BML gesture. Elckerlyc's BML scheduling algorithm is strictly separated from its BML parsing and multimodal plan management. This should allow one to easily replace Elckerlyc's default scheduling algorithm with a different one. All this functionality is achieved without requiring compile time dependencies of Elckerlyc (or the Behavior Planner) on the custom components that extend it. Other Realizers have implemented alternative and more elaborate scheduling algorithms, or provide motor control on modalities that are not present in Elckerlyc (e.g. blushing), or specialized behavior elements (e.g. walking). Elckerlyc's extensibility allows one to easily use such specialized behaviors on existing or new modalities. Most aspects of Elckerlyc's modular design are demonstrated in several virtual human applications and generic tools for testing or user experiments. The sole exception is Elckerlyc's ability to allow new scheduling algorithms; the feasibility of this design feature has yet to be demonstrated.

11.2 Designing a Virtual Human that Allows Continuous Interaction

Humans execute motor behavior (speech, gesture, facial expression, etc.) using several body parts. This movement of the body should not be seen as a process of executing a set of completely independent 'behaviors' steering separate body parts. Instead, behaviors are tightly coordinated. BML provides the means for the specification of both the behaviors that are to be executed, and their coordination (in the form of time constraints) between them. The coordination of behaviors is not limited to one's own body; tight coordination is observed between the 'behaviors' of interacting humans. I contribute the specification of such coordination using the BML extensions BML^{T}. Continuous interaction requires that a virtual human has a flexible behavior plan, which can be adjusted while it is being executed. Elckerlyc contributes such a flexible behavior plan representation.

In this thesis, I have highlighted the requirements of virtual humans to achieve tight coordination with the behavior of their interlocutor, both through a literature overview of continuous interaction in human-human communication and through requirement analysis of virtual human applications that already use some form of continuous interaction, or that require its use. Continuous interaction requires the specification of 1) the synchronization of (ongoing) behavior to predicted events originating from the environment or the virtual human's interlocutor, 2) instant interruption and fluent (alternative) continuation of ongoing behavior, 3) modifications in the shape of ongoing behavior, and 4) immediate execution of behavior, allowing apparent opportunistic planning.

The synchrony *between* the behavior of different humans relies on the very same synchronization mechanisms that exist between modalities (e.g. speech, gesture) *within* ones own body. Therefore, it makes sense to allow the specification of synchronization to behavior of an interlocutor in the same fashion as synchronization of modalities within ones own body. In BML, synchronization between behaviors of a virtual human on different modalities (e.g. speech, gesture) is specified as the alignment of the synchronization points of these behaviors. If synchronization to an interlocutor is to be handled in an analog fashion, it should be possible to specify synchronization of a virtual human's behavior to (predicted) synchronization points of the behavior of its interlocutor. In such a specification, multimodal behavior is described as part of a joint action, rather than as a completely autonomous act. I contribute BML^{T}, an extension of BML that satisfies the specification requirements of continuous interaction. It allows the specification of the synchronization to predicted interlocutor behavior, graceful interruption of ongoing behavior, the specification of parameter value changes in ongoing behavior and the pre-planning of behavior that can be instantly activated at a later stage. Elckerlyc can execute behavior specifications in BML^{T}.

In a Realizer that supports continuous interaction, the execution of behaviors should not be ballistic. Instead, it should be seen as a continuous process that offers rapid and fine-grained interruption and allows one to make microadjustments on the timing or shape of ongoing behaviors. These microadjustments in ongoing behavior should however not violate the timing constraints posed in the BML that constructed it. Elckerlyc contributes a flexible behavior plan representation that supports these requirements. Such a flexible plan requires PlanUnits that support microadjustments in timing and shape. I contribute an interface for such PlanUnits and provide several implementations for different types of behavior (gesture, speech, gaze, etc.).

Continuous interaction has consequences for the interpretation of the SAIBA planning 'pipeline'. To allow continuous interaction, this pipeline should not be seen as a unidirectional pipeline of intent planning, behavior planning and behavior realization. Instead, intent planning, behavior planning and behavior realization should be seen as parallel processes which *share* an ongoing behavior plan and incrementally modify and update it. I show that such a shared plan representation can already be achieved in the SAIBA framework, if SAIBA's feedback messages (communicating 'backward' in the pipeline) between the planning processes are

implemented.

11.3 Leveraging Computer Animation Knowledge for Interactive Virtual Human Applications

This thesis contributes a survey that provides an overview of naturalness and control trade-offs between different animation techniques. Choosing the right technique depends on the requirements of the application it is used in. Motion (capture) editing techniques employ the detail of captured motion or the talent of skilled animators, but they allow little deviation from the captured examples and can lack physical realism. Procedural motion offers detailed and precise control using a large number of parameters, but lacks naturalness. Physical simulation provides integration with the physical environment and physical realism. However, physical realism alone is not enough for naturalness and physical simulation offers poor precision in both timing and limb placement. Hybrid animation techniques combine and concatenate motion generated by different animation paradigms to enhance both naturalness and control.

Elckerlyc has pioneered the use of physical simulation in a real-time multimodal virtual human platform. It combines the physical naturalness provided by physically realistic animation with the control provided by procedural animation. To this end it contributes a hybrid animation technique: mixed dynamics. It builds on the notion that the requirements of physical integrity and tight temporal synchronization are often of different importance for different body parts. For example, for a gesturing virtual human, tight synchronization with speech is primarily important for arm and head movement. At the same time, a physically valid balancing motion of the whole body could be achieved by moving only the lower body, where precise timing is less important. Mixed dynamics allows one to mix procedural arm and head gestures with physical simulation of the rest of the body. The forces generated by the gesturing body parts are transfered to the physically simulated body parts, thus creating whole body animation that appears to respect the laws of physics in a believable manner and that is internally coherent (that is: the movement of the physically steered body parts is affected by the movement of the procedurally steered ones). This provides physically coherent whole body involvement, a naturalness feature that is lacking in virtual human platforms that solely use procedural animation.

Chapter 12

Discussion

Interactions between humans are characterized by continuous interpersonal coordination. Interactants align their behaviors in form, content or timing. In this thesis, I have focused on allowing the coordination of form and timing. Interpersonal coordination can be categorized in behavior matching and interactional synchrony. Interactional synchrony includes alignment of movement rhythm, synchronization of behavior between interlocutors and smooth meshing/intertwining of behavior. Elckerlyc is the first multimodal virtual behavior generation platform that has been designed to support all of these aspects of interactional synchrony.

Other work has focused on several other aspects of interpersonal coordination. Recent work on Behavior Planners provides the *alignment of content and form* (and thus behavior matching) in virtual humans [29, 43, 97, 116]. Some other multimodal synthesis systems provide *incremental synthesis* algorithms specialized for the generation of gesture and speech [105, 155]. These algorithms allow gesture co-articulation, which currently cannot be easily expressed in BML. Several input systems allowing continuous interaction have recently been developed. These systems provide *incremental and predictive input processing*. Such incremental output generation and input prediction contributes to achieving smooth meshing and intertwining of behavior with that of the interlocutor.

To achieve interpersonal coordination, a virtual human should be able to exhibit behavior matching and interactional synchrony. In this Chapter, I illustrate how my work might be integrated with the efforts in behavior planning, behavior realization and input processing mentioned above to design a virtual human that truly exhibits interpersonal coordination.

12.1 Gesture Co-Articulation in a BML Realizer

Some non-BML[1] virtual human platforms have implemented scheduling algorithms that are specialized for scheduling synchronized speech and gesture. These algorithms allow biologically inspired gesture co-articulation. For example: they will

[1]The authors of these systems are part of the SAIBA initiative. It is their intent to make their platforms BML compliant in the future.

skip the retraction of a gesture if it is immediately followed by another gesture. In this Section I illustrate how this co-articulation is implemented in Greta and Max. I discuss the required additions to BML to allow the specification of gesture co-articulation in incremental speech-gesture production and illustrate how it could be implemented in Elckerlyc.

12.1.1 Greta: Speech-aligned Scheduling

In Greta's scheduling algorithm [105], gesture timing is solely determined by speech timing. The end of the gesture stroke is aligned with the emphasized word. The retraction and preparation phases of gestures can be stretched or skewed. The maximum stretch and skew of preparation and retraction are annotated with each gesture. If the preparation phase of a gesture is to be skewed beyond its minimum, the gesture is dropped. If an extended period of time separates two gestures, a rest pose is inserted between them. Using this mechanism scheduling is completely deterministic and all time constraints can be determined beforehand.

12.1.2 Max: Chunk-based Scheduling

The Max system [155] is the first virtual human system that simulates the *mutual* adaptations between the timing of gesture and speech that humans employ to achieve synchrony between the co-expressive elements in those two modalities. It also pioneers the incremental scheduling of multimodal behavior for virtual humans.

Max's incremental speech-gesture production model is based on McNeill's segmentation hypothesis [189]: speech and gesture are produced in successive *chunks*. Each chunk contains one tone unit in speech and one co-expressive gesture phrase. Within each chunk, the prominent concept is concertedly conveyed by a gesture and an affiliated word or sub phrase (in short, affiliate). Each tone unit has exactly one primary pitch accent, which is called the *nucleus*.[2]

The gesture phrase is aligned to the tone unit in such a way that the stroke phase of the gesture starts before (in Max this is 0.3s or one syllable) the affiliate in speech and frequently spans it, optionally by inserting a dedicated hold phase. If one of the affiliated words is prosodically focused (for emphasizing/contrasting purposes) the gesture stroke starts exactly at the nucleus.

Gesture movement between the successive strokes of two gestures (in two successive chunks) depends on their timing, it may range from moving to an inbetween rest position to a direct transition movement. A silence might need to be inserted between the tone units in successive chunks, depending on the duration of the preparation phase of the second gesture. To achieve this production flexibility, Max uses an incremental scheduling algorithm that plans part of the chunk in advance. It

[2]See Chapter 4.3.2 for a more extensive overview of the hierarchical composition and synchronization of gesture and speech.

refines this plan at a later stage (when the chunk is actually started) by setting up each chunk's inter-chunk synchrony with its predecessor (see Inset 1).

The incremental speech-gesture scheduling algorithm in Max [155] plans speech and gesture as sequences of chunks containing one gesture phrase and one tone unit. The chunks move through the following phases:

- *InPrep* The speech synthesis system synthesizes the tone unit and provides its timing at phoneme level. The gesture planner selects a lexicalized gesture template, allocates body parts, expands abstract movement constraints and resolves deictic references.
- *Pending* Set once InPrep is completed
- *Lurking* If a chunk can be uttered, i.e. when the preceding chunk is subsiding, the scheduler sets up the inter-chunk synchrony between the two chunks.
- *InExec* Set once Lurking is completed, executes the chunk.
- *Subsiding* Retraction phase of the gesture, speech finished.
- *Done* Finished executing the chunk.

Note that new chunks are prepared while their preceding chunk is still executing.

Inset 1: The scheduling algorithm used in Max.

12.1.3 Specifying Chunk-based Concatenation in BML

Chunk based scheduling requires that the multimodal behavior plan is built up incrementally, with each chunk potentially modifying the timing and execution of the previous chunk (e.g. by skipping the retraction phase in its gesture). A chunk starts when the previous chunk is subsiding, that is, when its gesture is in or past the retraction phase and the speech is finished. BML blocks are the incremental blocks that construct the behavior plan of a Realizer. It would therefore make sense to specify each chunk in a BML block (as in BML Example 32).

BML Example 32 A chunk expressed in BML.

```
<bml id="bml1">
  <speech id="speech1"><text>... <sync id="sync1"> ... </text></speech>
  <gesture id="g1" type=.. stroke="speech1:sync1"/>
</bml>
```

However, BML does not currently allow relations between blocks other than appending one block at the end of another block. A natural way to extend the synchronization between BML blocks is to allow a block to synchronize to synchronization

points of another block and thus to define synchronization points of the BML block itself. The start, ready, relax and end of a BML block could be defined as follows:

- start: start of the first behavior in the BML block
- ready: ready of the first behavior in the BML block
- relax: relax of the last behavior in the BML block
- end: end of the last behavior in the BML block

BML Example 33 shows how the `bml2` is started at the relax phase of `bml1`, thus providing a concatenation of BML blocks that is similar to chunk based concatenation.

BML Example 33 Synchronizing the start of BML Block `bml2` to the relax of `bml1`.

```
<bml id="bml2" start="bml1:relax">
  ...
</bml>
```

Note that this notation can also express the append-after (but not the append) scheduling attribute (see BML Example 34).

BML Example 34 This constraint requires `bml4` to start after `bml1`, `bml2` and `bml3`.

```
<bml id="bml4">
  <constraint id="s1">
    <before ref="bml4:start">
      <sync ref="bml1:end"/>
      <sync ref="bml2:end"/>
      <sync ref="bml3:end"/>
    </before>
  </constraint>
  ...
</bml>
```

Potentially, it can also be used to *define* the sync points of a BML block in relation to its internal behavior (see BML Example 35).

BML Example 35 Defining the stroke of `bml1` as the stroke of its behavior `g1`.

```
<bml id="bml1" stroke="g1:stroke">
  <speech id="speech1"><text>... <sync id="sync1"> ... </text></speech>
  <gesture id="g1" type=.. stroke="speech1:sync1"/>
</bml>
```

12.1.4 Chunk-based Scheduling in Elckerlyc

In addition to requiring the start sync of one BML block to synchronize with the relax sync of another, chunk-based scheduling requires a flexible adaptation of the timing of the next BML block (which is already partly scheduled). The timing of the next BML block is dependent on the duration of the preparation phase of the gesture in this BML block, which is in turn dependent on the position of, for example, the hand at the relax sync of the previous BML block. This hand position (and thus this timing) is known as soon as the BML block starts. Just before the chunk starts, the timing of the stroke of the gesture (and thus the connected nucleus in speech) should thus be adjusted in such a way that the gesture has the preferred preparation duration. Scheduling a chunk in Elckerlyc results in a behavior plan in which the nucleus of the speech in the chunk and the stroke of the gesture in the chunk are connected to the same TimePeg (see Figure 12.1).

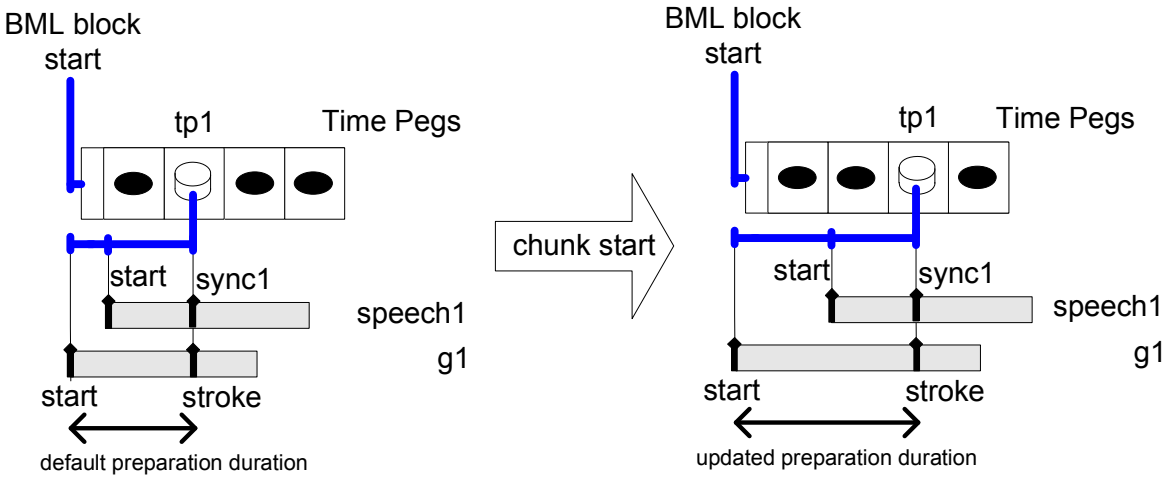


Figure 12.1: Updating the timing of a scheduled chunk to set a preferred preparation duration.

The start sync of the gesture (determined by its default preparation duration) and the start sync of the speech segment are both connected with an OffsetPeg to this TimePeg. Before the chunk is started, the desired preparation duration of the gesture can be achieved by modifying the timing of both this TimePeg and the time offset of the OffsetPeg of the gesture (see Figure 12.1). Elckerlyc's behavior plan supports the application of such timing modifications. Implementing chunk based scheduling in Elckerlyc thus entails implementing some mechanism that, at the appropriate time, applies this timing modification to a chunk. The exact form of this 'mechanism' and its combination with/generalization to BML blocks that are not chunks is an open research question.

12.2 Continuous Input

Elckerlyc provides continuous output generation. In most interactive applications using Elckerlyc input handling was achieved using a Wizard of Oz setup. An au-

tonomous virtual human that allows continuous interaction with its interlocutors requires not only continuous and incremental output generation but also continuous and incremental input processing. This input processing is often required to provide not only the detection, but also the prediction of interlocutor and/or world events. The Reactive Virtual Trainer is a prime example of an autonomous virtual human application using Elckerlyc's continuous interaction capabilities. It makes use of predictions of the user's movement using the ExcerciseAnticipator. Recently, progress has been made in incremental and/or predictive input processing in conversational settings. Using this work, Anticipators could be implemented that allow the design of a virtual human that uses Elckerlyc's continuous interaction capabilities autonomously within a conversation. This section illustrates some of the state-of-the-art work on incremental and/or predictive input processing that can be applied in a conversational setting.

12.2.1 Incremental Speech Recognition

Traditionally, automatic speech recognition systems are used to recognize speech in segments containing roughly one sentence at a time. Many speech recognizers, including Sphinx-4 [300] and SHoUT[3] may be used in an incremental mode. In such an incremental mode, the speech recognizer provides a new hypothesis of the words spoken after each input frame (often every 10ms) [21]. Bauman et al. [21] shows that incremental speech recognition allows prediction, that is, the first hypothesis on a word becomes available before the speaker has finished speaking it.

12.2.2 End-of-Turn Prediction

Traditionally, spoken dialog systems have used pause length (between 0.5s and 1s) as a cue for taking the turn [252].[4] In human-human conversation, turn-taking is much more fluent: one interaction participant starts speaking immediately after (or even before) the previous speaker finishes his turn (as discussed in Chapter 5.1.3). Schlangen argues that to achieve such more human-like and fluent turn-taking, prediction of rather than reaction to the end of turn is required. He provides a classifier that uses prosodic and syntactic information that detects whether a certain word in speech is the end of a turn or not. Later work improves the performance of the classifier [12]. De Kok and Heylen [147] contribute an end-of-turn predictor that uses a multimodal set of annotated input features in a multi-party meeting. Their end of turn predictor makes use of dialogue acts, focus of attention, head gestures and prosody.

Jonsdottir et al. [130] contribute a talking agent that can learn fluent turn-taking behavior *during* an interaction. Using online machine learning with prosodic

[3] http://shout-toolkit.sourceforge.net/

[4] A notable exception is the Rea system [48] which uses a combination of detected phrase type (imperative, interrogative of declarative), user gesture and (when needed, depending on phrase type) pause length as turn-taking cues. This effectiveness of this approach has never been evaluated [130].

features as input, the agent learns to time the start of its turn in such a way that silence between turns is minimized. The effectiveness of this approach is demonstrated by making the agent talk with a copy of itself. In later work, the agent is employed as an interviewer that interviews a human interlocutor [129].

12.2.3 Listener Response Relevant Moment Prediction

To allow a virtual human to provide listener responses at the appropriate time, it should be able to predict, or at least detect, the relevant moments to do so during an ongoing utterance from its interlocutor. Real-time detectors for relevant moments to generate *generic* listener responses have been developed using surface features of speech and gaze. J. Jr [282] and Rea [48] determine appropriate moments to generate generic listener responses on the basis of silence durations of ongoing utterances. Maatman et al. [182] provide a more elaborate model to detect relevant moments for generic listener responses. It makes use of both pause durations in and acoustic features of ongoing speech. Morency et al. [197] provide a multimodal detector for relevant moments for generic listener responses, making use of gaze, prosody, words uttered and pause durations. Jonsdottir et al. [128] contribute a system that can detect appropriate moments for *specific* listener feedback (and generates them), within a limited domain. It selects appropriate specific listener responses and response moments on the basis of keywords spotted in ongoing speech.

12.2.4 Online Speech Classification

When determining how to respond to incoming speech from a user, it is crucial that a virtual human knows whether the incoming user utterance is a listener response or some other speech act. Most listener responses are generic listener responses [23]. When receiving such a generic listener response, it is typically appropriate for the virtual human to simply continue speaking (see also Chapter 5.1.4). Specific listener responses or other utterances from a user often require retiming, reparameterization or interruption of ongoing behavior. Neiberg and Truong [208] provide a classifier that can, on the basis of acoustic features of online speech, determine whether incoming speech is a listener response or not. This classifier can be configured to provide a classification result after a maximum latency. This latency can be adjusted to provide a trade-off between responsivity and classification accuracy. In a virtual human system that allows incremental output, it can be useful to run multiple classifiers (each with a different latency) in parallel. This allows the virtual human to generate behavior based on early classifications, which can —if necessary— be modified, when a later and more accurate classification becomes available.

12.3 Interactional Synchrony

Elckerlyc is the first multimodal virtual human generation platform that is designed to support all aspects of interactional synchrony: interpersonal behavior synchronization, rhythmic alignment and smooth meshing of behavior with that of an interlocutor. This is achieved by 1) support for the specification of behavior alignment to predicted and changing time events from the interlocutor, 2) a flexible multimodal behavior plan representation that allows plan modification while retaining the constraints defined within it, and 3) flexible PlanUnits within the behavior plan that allowing changes to their timing even while they are being executed.

Elckerlyc's interpersonal behavior synchronization capabilities are currently used in the Reactive Virtual Trainer, to synchronize exercise motion with that of a user. Here, interactive synchrony entails synchronizing key time moments in an exercise motion to predicted key time moments of a user. Similar synchronization can be used in a conversation, for example for interpersonal synchronization of gestures. Rhythmic alignment, for example the entrainment of postural sway, does not require synchronization of *specific* time moments in specific behaviors, but might be better described with synchronization tendency parameters (in-phase, anti-phase, variability, etc.). Currently Elckerlyc does not contain the *specification* mechanisms to handle this kind of synchronization. However, Elckerlyc's multimodal behavior plan and its PlanUnits are flexible enough to be able to handle the behavior execution and on-the-fly plan modification required for it.

12.4 Towards Interpersonal Coordination with Virtual Humans

Interactions between humans are characterized by continuous interpersonal coordination. Interactants assimilate their behaviors in form, content or timing. Elckerlyc focuses on allowing the coordination of form and timing. Other work on interpersonal coordination using virtual humans has focused on the alignment of content and form [29, 43, 97, 116], which is typically achieved at the SAIBA Behavior Planner level. This includes behavior matching, for example the alignment of lexical items, the alignment of syntactic structures, the alignment in gesture use or unconscious mimicry. These approaches can provide behavior plans (in BML) that can be executed by Elckerlyc. Elckerlyc thus complements the interpersonal coordination capabilities of these approaches, contributing form and timing matching at Realizer level.

Several interpersonal coordination phenomena require the alignment of behavior to *predicted* events of interlocutor behavior. Furthermore, to allow timely output generation, incremental input processing is required. As discussed in Section 12.2, recent work has contributed several incremental and/or predictive input processing modules that can be used in the context of conversations with a virtual human.

A virtual human that truly exhibits interpersonal coordination, including behavior matching, interactional synchrony (including interpersonal behavior synchrony,

rhythmic alignment, and smooth meshing and intertwining of behavior with that of the interlocutor) can potentially be designed using a combination of all work mentioned above. The exact execution (e.g. the timing, the shape and the amount) of interaction coordination can influence the perceived personality and emotional state of a (virtual) human in subtle, context dependent ways. To achieve natural interaction, one should thus not aim for a virtual human that exhibits as much coordination with its interaction partner as possible, but rather for coordinative behavior that matches the virtual human's personality, its emotional state, the current interaction context, and so on. A remaining interesting question is then how such parameters should exactly configure the interactional coordination that is applied to a virtual human.

Bibliography

[1] Y. Abe and J. Popović. Interactive animation of dynamic manipulation. In *SCA '06: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 195–204. Eurographics Association, Aire-la-Ville, Switzerland, 2006.

[2] Y. Abe, M. da Silva, and J. Popović. Multiobjective control with frictional contacts. In *SCA '07: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 249–258. Eurographics Association, Aire-la-Ville, Switzerland, 2007.

[3] S. V. Adamovich, M. F. Levin, and A. G. Feldman. Merging different motor patterns: Coordination between rhythmical and discrete single-joint movements. *Experimental Brain Research*, 99(2):325–337, 1994.

[4] A. A. H. Ahmed. *Parametric Synthesis of Human Animation*. Ph.D. thesis, University of Surrey, Surrey, UK, April 2004.

[5] R. op den Akker, S. Creer, M. ter Maat, D. Reidsma, P. Wallis, and J. Zwiers. SERA Deliverable 3.3: Implementation of the SERA Core Architecture, 2011.

[6] B. Allen, D. Chu, A. Shapiro, and P. Faloutsos. On the beat!: timing and tension for dynamic characters. In *SCA '07: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 239–247. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2007.

[7] J. Allwood and L. Cerrate. A study of gestural feedback expressions. In P. Paggio, K. Jokinen, and K. Jönsson, editors, *1st Nordic Symposium on Multimodal Communication*, pages 7–22. September 2003.

[8] K. Amaya, A. Bruderlin, and T. Calvert. Emotion from motion. In *Proceedings of Graphics Interface*, pages 222–229. Canadian Information Processing Society, Toronto, Ontario, Canada, May 1996.

[9] E. André, T. Rist, S. van Mulken, M. Klesen, and S. Baldes. The automated design of believable dialogues for animated presentation teams. In J. Cassell, J. Sullivan, S. Prevost, and E. F. Churchill, editors, *Embodied conversational agents*, pages 220–255. MIT Press, Cambridge, MA, USA, 2000.

[10] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *ACM Transactions on Graphics*, volume 21, pages 483–490. ACM, New York, NY, USA, 2002.

[11] Artificial Life Solutions. Massive Prime, 2008. Http://www.massivesoftware.com/prime/.

[12] M. Atterer, T. Baumann, and D. Schlangen. Towards Incremental End-of-Utterance Detection in Dialogue Systems. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 11–14. 2008.

[13] N. I. Badler, J. Allbeck, L. Zhao, and M. Byun. Representing and Parameterizing Agent Behaviors. In *Proceedings of Computer Animation*, pages 133–143. IEEE Computer Society, Washington, DC, USA, 2002.

[14] J. Bailenson and N. Yee. Digital chameleons: Automatic assimilation of nonverbal gestures in immersive virtual environments. *Psychological Science*, 16(10):814–819, October 2005.

[15] K. Balci. Xface: Open Source Toolkit for Creating 3D Faces of an Embodied Conversational Agent. In A. Butz, B. Fisher, A. Krüger, and P. Olivier, editors, *Proceedings of the 5th International Symposium on Smart Graphics*, volume 3638 of *Lecture Notes in Computer Science*, pages 924–924. Springer Berlin / Heidelberg, 2005.

[16] P. Barkhuysen, E. Krahmer, and M. Swerts. The interplay between the auditory and visual modality for end-of-utterance detection. *Journal of the Acoustical Society of America*, 123(1):354–365, 2008.

[17] R. Barzel, J. F. Hughes, and D. N. Wood. Plausible motion simulation for computer graphics animation. In R. Boulic and G. Hégron, editors, *Proceedings of the Eurographics workshop on Computer Animation and Simulation*, pages 183–197. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[18] B. J. H. van Basten and A. Egges. Evaluating distance metrics for animation blending. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 199–206. ACM, New York, NY, USA, 2009.

[19] B. J. H. van Basten, P. W. A. M. Pieters, and A. Egges. The step space: example-based footprint-driven motion synthesis. *Computer Animation and Virtual Worlds*, 21(3), 2010.

[20] B. J. H. van Basten, S. A. Stüvel, and A. Egges. Exact Parameterization for Footprint-driven Synthesis. In *Poster Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 7–8. ACM, New York, NY, USA, 2010.

[21] T. Baumann, M. Atterer, and D. Schlangen. Assessing and improving the performance of speech recognition for incremental systems. In *NAACL '09: Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 380–388. Association for Computational Linguistics, Stroudsburg, PA, USA, 2009.

[22] J. B. Bavelas, A. Black, C. R. Lemery, and J. Mullet. I Show How You Feel: Motor Mimicry as a Communicative Act. *Journal of Personality and Social Psychology*, 50(2):322–329, 1986.

[23] J. B. Bavelas, L. Coates, and T. Johnson. Listeners as co-narrators. *Journal of Personality and Social Psychology*, 79(6):941–952, 2000.

[24] J. B. Bavelas, L. Coates, and T. Johnson. Listener responses as a collaborative process: The role of gaze. *Journal of Communication*, 52(3):566–580, 2002.

[25] K. Beck. *Extreme Programming Explained*. Addison Wesley, Reading, MA, USA, 1999.

[26] E. Berglund and M. Priestley. Open-source documentation: in search of user-driven, just-in-time writing. In *SIGDOC '01: Proceedings of the 19th annual international conference on Computer documentation*, pages 132–141. ACM, New York, NY, USA, 2001.

[27] F. J. Bernieri. Coordinated movement and rapport in teacher-student interactions. *Journal of Nonverbal Behavior*, 12:120–138, 1988.

[28] F. J. Bernieri and R. Rosenthal. Interpersonal coordination: Behavior matching and interactional synchrony. In R. S. Feldman and B. Rimé, editors, *Fundamentals of Nonverbal Behavior*, Studies in Emotional and Social Interaction, chapter 11. Cambridge University Press, 1991.

[29] E. Bevacqua, E. Prepin, R. de Sevin, R. Niewiadomski, and C. Pelachaud. Reactive Behaviors in SAIBA Architecture. In *Workshop Towards a Standard Markup Language for Embodied Dialogue Acts, held in conjunction with AAMAS'09*. 2009.

[30] T. W. Bickmore, L. M. Pfeifer, and M. K. Paasche-Orlow. Health Document Explanation by Virtual Agents. In C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis, and D. Pelé, editors, *Proceedings of the 7th international conference on Intelligent Virtual Agents*, volume 4722 of *Lecture Notes in Computer Science*, pages 183–196. Springer-Verlag, Berlin, Heidelberg, 2007.

[31] E. Bizzi, A. D'Avella, P. Saltiel, and M. Tresch. Modular organization of spinal motor systems. *Neuroscientist*, 8(5):437–442, 2002.

[32] J. Blascovich, J. Loomis, A. Beall, K. Swinth, C. Hoyt, and J. Bailenson. Immersive virtual environment technology as a methodological tool for social psychology. *Psychological Inquiry*, 13(2):103–124, 2002.

[33] B. Bodenheimer, A. V. Shleyfman, and J. K. Hodgins. The Effects of Noise on the Perception of Animated Human Running. In N. Magnenat-Thalmann and D. Thalmann, editors, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, pages 53–63. September 1999.

[34] M. Bodewitz. *Detecting the student personality*. Master's thesis, University of Twente, 2004.

[35] A. Boeing and T. Bräunl. Evaluation of real-time physics simulation systems. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 281–288. ACM, New York, NY, USA, 2007.

[36] S. M. Boker, J. F. Cohn, B.-J. Theobald, I. Matthews, T. R. Brick, and J. R. Spies. Effects of damping head movement and facial expression in dyadic conversation using realtime facial expression tracking and synthesized avatars. *Philosophical Transactions of the Royal Society*, 364(1535):3485–3495, 2009.

[37] R. Boulic and R. Kulpa. Inverse Kinematics and Kinetics for Virtual Humanoids. In *Eurographics Tutorials*, T4, pages 643–742. 2007.

[38] M. Brand and A. Hertzmann. Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[39] H. P. Branigan, M. J. Pickering, J. Pearson, and J. F. McLean. Linguistic alignment between people and computers. *Journal of Pragmatics*, 42(9):2355 – 2368, 2010. How people talk to Robots and Computers.

[40] L. C. Briand. Software Documentation: How Much Is Enough? In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, pages 13–15. IEEE Computer Society, Washington, DC, USA, 2003.

[41] A. Bruderlin and L. Williams. Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104. ACM Press, New York, NY, USA, 1995.

[42] K. Brügmann, H. Dohrn, H. Prendinger, M. Stamminger, and M. Ishizuka. Phase-based gesture motion parametrization and transitions for conversational agents with MPML3D. In *Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*, pages 1–6. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels, Belgium, 2008.

[43] H. Buschmeier, K. Bergmann, and S. Kopp. Adaptive expressiveness: virtual conversational agents that can align to their interaction partner. In *AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 91–98. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2010.

[44] B. L. Callennec and R. Boulic. Interactive motion deformation with prioritized constraints. In *SCA '04: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 163–171. Eurographics Association, 2004.

[45] R. H. S. Carpenter. *Movements of the Eyes*. Pion Ltd, London, UK, second edition, 1988.

[46] S. R. Carvalho, R. Boulic, and D. Thalmann. Interactive low-dimensional human motion synthesis by combining motion models and PIK. *Computer Animation and Virtual Worlds*, 18(4-5):493–503, 2007.

[47] J. Cassell. Body Language: Lessons from the Near-Human. In J. Riskin, editor, *Genesis redux*, chapter 17, pages 346–374. University Of Chicago Press, 2007.

[48] J. Cassell, T. W. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. H. Vilhjálmsson, and H. Yan. Embodiment in conversational interfaces: Rea. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 520–527. ACM, New York, NY, USA, 1999.

[49] J. Cassell, Y. I. Nakano, T. W. Bickmore, C. L. Sidner, and C. Rich. Annotating and Generating Posture from Discourse Structure in Embodied Conversational Agents. In *Proceedings of the Workshop on Representating, Annotating, and Evaluating Non-Verbal and Verbal Communicative Acts to Achieve Contextual Embodied Agents at Autonomous Agents*. 2001.

[50] J. Cassell, C. Pelachaud, N. I. Badler, M. Steedman, B. Achorn, T. Becket, B. Douville, S. Prevost, and M. Stone. Animated conversation: rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 413–420. ACM, New York, NY, USA, 1994.

[51] J. Cassell, J. Sullivan, S. Prevost, and E. F. Churchill. *Embodied Conversational Agents*. The MIT Press, Cambridge, MA, USA, 2000.

[52] J. Cassell and K. R. Thórisson. The Power of a Nod and a Glance: Envelope vs. Emotional Feedback in Animated. *Applied Artificial Intelligence*, 13(4-5):519–538, 1999.

[53] J. Cassell, H. H. Vilhjálmsson, and T. W. Bickmore. BEAT: the Behavior Expression Animation Toolkit. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 477–486. ACM, New York, NY, USA, 2001.

[54] A. Čereković and I. S. Pandžić. Multimodal behavior realization for embodied conversational agents. *Multimedia Tools and Applications*, pages 1–22, 2010.

[55] T. Chaminade, J. K. Hodgins, and M. Kawato. Anthropomorphism influences perception of computer-animated characters' actions. *Social Cognitive and Affective Neuroscience*, 2(3):206–216, May 2007.

[56] S.-P. Chao, S.-N. Yang, and T.-G. Lin. An LMA-Effort simulator with dynamics parameters for motion capture animation. *Computer Animation and Virtual Worlds*, 17(3-4):167–177, 2006.

[57] T. L. Chartrand and J. A. Bargh. The chameleon effect: the perception-behavior link and social interaction. *Journal of Personality and Social Psychology*, 79(6), 1999.

[58] D. M. Chi, M. Costa, L. Zhao, and N. I. Badler. The EMOTE Model for Effort and Shape. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 173–182. ACM Press/Addison-Wesley Publishing Co., 2000.

[59] H. H. Clark. *Using language*. Cambridge University Press, Cambridge, UK, 1996.

[60] W. S. Condon and M. B. Ogston. Sound Film Analysis of Normal and Pathological Behavior Patterns. *Journal of Nervous and Mental Disease*, 143(4):338–347, 1966.

[61] S. Coros, P. Beaudoin, and M. van de Panne. Robust task-based control policies for physics-based characters. *ACM Transactions on Graphics*, 28(5):170:1–170:9, 2009.

[62] E. Coumans. Bullet Open Source Physics Library, 2008. Http://www.bulletphysics.com/.

[63] M. Courgeon. The MARC framework, 2011. Http://marc.limsi.fr/.

[64] M. Courgeon, J.-C. Martin, and C. Jacquemin. MARC: a Multimodal Affective and Reactive Character. In *Proceedings of the 1st Workshop on AFFective Interaction in Natural Environements*. 2008.

[65] B. De Carolis, C. Pelachaud, I. Poggi, and M. Steedman. APML, a Markup Language for Believable Behavior Generation. In *Life-Like Characters: Tools, Affective Functions, and Applications*, pages 65–86. Springer, January 2004.

[66] E. Dehling. *The Reactive Virtual Trainer*. Master's thesis, University of Twente, Enschede, the Netherlands, 2011.

[67] W. T. Dempster and G. R. L. Gaughran. Properties of Body Segments Based on Size and Weight. *American Journal of Anatomy*, 120(1):33–54, 1967.

[68] A. Egges and N. Magnenat-Thalmann. Emotional Communicative Body Animation for Multiple Characters. In *Workshop on Crowd Simulation*, pages 31–40. November 2005.

[69] A. Egges, T. Molet, and N. Magnenat-Thalmann. Personalised Real-Time Idle Motion Synthesis. In *PG '04: Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, pages 121–130. IEEE Computer Society, Washington, DC, USA, 2004.

[70] A. Egges, R. M. Visser, and N. Magnenat-Thalmann. Example-Based Idle Motion Synthesis in a Real-Time Application. In *CAPTECH Workshop*, pages 13–19. December 2004.

[71] P. Ekman and W. Friesen. *Facial Action Coding System: A Technique for the Measurement of Facial Movement.* Consulting Psychologists Press, Palo Alto, CA, UK, 1978.

[72] C. Elliott and P. Hudak. Functional reactive animation. In *ICFP '97: Proceedings of the 2nd ACM SIGPLAN international conference on Functional programming*, pages 263–273. ACM, New York, NY, USA, 1997.

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

[74] A. Fang and N. S. Pollard. Efficient Synthesis of Physically Valid Human Motion. *ACM Transactions on Graphics*, 22(3):417–426, July 2003.

[75] R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer, New York, NY, USA, October 2007.

[76] A. G. Feldman. Once more on the equilibrium-point hypothesis (lambda model) for motor control. *Journal of Motor Behaviour*, 18(1):17–54, March 1986.

[77] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

[78] T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of Neuroscience*, 5(7):1688–1703, July 1985.

[79] D. A. Forsyth, O. Arikan, L. K. M. Ikemoto, J. F. O'Brien, and D. Ramanan. Computational studies of human motion: part 1, tracking and motion synthesis. *Foundation and Trends in Computer Graphics Vision*, 1(2):77–254, 2006.

[80] D. T. Fujimoto. Listener Responses in Interaction: A Case for Abandoning the Term, Backchannel. *Journal of Osaka Jogakuin 2 year College*, 37:35–54, 2007.

[81] N. Furuyama. Prolegomena of a theory of between-person coordination of speech and gesture. *International Journal of Human-Computer Studies*, 57(4):347 – 374, 2002.

[82] N. Furuyama, D. McNeill, and M. Park-Doob. Is speech-gesture production ballistic or interactive? In *First Congress of the International Society for Gesture Studies*. University of Texas, June 2002.

[83] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Adisson-Wesley, Reading, MA, USA, 1995.

[84] S. Garrod and M. J. Pickering. Why is conversation so easy? *Trends in Cognitive Sciences*, 8(1):8–11, 2004.

[85] J. B. Gatewood and R. Rosenwein. Interactional synchrony: Genuine or spurious? A critique of recent research. *Journal of Nonverbal Behavior*, 6(1):12–29, 1981.

[86] S. Gibet, J.-F. Kamp, and F. Poirier. Gesture Analysis: Invariant Laws in Movement. In *Gesture-Based Communication in Human-Computer Interaction*, volume 2915 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin / Heidelberg, April 2004.

[87] S. Gibet, T. Lebourque, and P.-F. Marteau. High-level Specification and Animation of Communicative Gestures. *Journal of Visual Languages & Computing*, 12(6):657–687, 2001.

[88] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, NY, USA, January 1982.

[89] P. Glardon, R. Boulic, and D. Thalmann. Robust on-line adaptive footplant detection and enforcement for locomotion. *The Visual Computer*, 22(3):194–209, 2006.

[90] M. Gleicher. Motion editing with spacetime constraints. In *SI3D '97: Proceedings of the symposium on Interactive 3D graphics*, pages 139–148. ACM Press, New York, NY, USA, 1997.

[91] M. Gleicher. Comparing constraint-based motion editing methods. *Graphical Models*, 63(2):107–134, 2001.

[92] M. Gleicher. More Motion Capture in Games - Can We Make Example-Based Approaches Scale? In A. Egges, A. Kamphuis, and M. Overmars, editors, *Motion in Games*, volume 5277 of *Lecture Notes in Computer Science*, pages 82–93. Springer-Verlag, 2008.

[93] M. Gleicher, H. J. Shin, L. Kovar, and A. Jepsen. Snap-together motion: assembling run-time animations. In *I3D '03: Proceedings of the symposium on Interactive 3D graphics*, pages 181–188. ACM, New York, NY, USA, 2003.

[94] J. A. Goldberg. Interrupting the discourse on interruptions : An analysis in terms of relationally neutral, power- and rapport-oriented acts. *Journal of Pragmatics*, 14(6):883 – 903, 1990.

[95] C. Goodwin. Between and within: Alternative sequential treatments of continuers and assessments. *Human Studies*, 9(2-3):205–217, 1986.

[96] F. S. Grassia. *Believable automatically synthesized motion by knowledge-enhanced motion transformation*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2000.

[97] J. Gratch, A. Okhmatovskaia, F. Lamothe, S. C. Marsella, M. Morales, R. van der Werf, and L.-P. Morency. Virtual Rapport. In J. Gratch, M. Young, R. Aylett, D. Ballin, and P. Olivier, editors, *Proceedings of the 6th International Conference on Intelligent Virtual Agents*, volume 4133 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2006.

[98] J. Gratch, J. W. Rickel, E. Andre, J. Cassell, E. Petajan, and N. I. Badler. Creating interactive virtual humans: some assembly required. *Intelligent Systems, IEEE*, 17(4):54–63, Jul/Aug 2002.

[99] A. Gravano and J. Hirschberg. Backchannel-Inviting Cues in Task-Oriented Dialogue. In *Proceedings of Interspeech*, pages 1019–1022. Brighton, 2009.

[100] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. *ACM Transactions on Graphics*, 23(3):522–531, 2004.

[101] D. Ha and J. Han. Motion synthesis with decoupled parameterization. *The Visual Computer*, 24(7):587–594, 2008.

[102] H. Haken, J. A. S. Kelso, and H. Bunz. A theoretical model of phase transitions in human hand movements. *Biological Cybernetics*, 51(5):347–356, 1985.

[103] C. M. Harris and D. M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394:780–784, 1998.

[104] B. Hartmann, M. E. Leventon, M. Mancini, W. T. Freeman, and C. Pelachaud. Implementing Expressive Gesture Synthesis for Embodied Conversational Agents. In S. Gibet, N. Courty, and J.-F. Kamp, editors, *Gesture in Human-Computer Interaction and Simulation*, volume 3881 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2005.

[105] B. Hartmann, M. Mancini, and C. Pelachaud. Formational Parameters and Adaptive Prototype Instantiation for MPEG-4 Compliant Gesture Synthesis. In *Proceedings of Computer Animation*, pages 111–119. IEEE Computer Society, Washington, DC, USA, 2002.

[106] Havok. Havok Physics, 2008. Http://www.havok.com/.

[107] R. Heck and M. Gleicher. Parametric Motion Graphs. In *I3D '07: Proceedings of the symposium on Interactive 3D graphics and games*, pages 129–136. ACM, New York, NY, USA, 2007.

[108] R. Heck, L. Kovar, and M. Gleicher. Splicing Upper-Body Actions with Locomotion. *Computer Graphics Forum*, 25(3), September 2006.

[109] R. M. Heck. *Automated authoring of quality human motion for interactive environments*. Ph.D. thesis, University of Wisconsin at Madison, 2007.

[110] M. Heldner and J. Edlund. Pauses, gaps and overlaps in conversations. *Journal of Phonetics*, 38(4):555–568, 2010.

[111] A. Heloir and M. Kipp. Real-Time Animation of Interactive Agents: Specification and Realization. *Applied Artificial Intelligence*, 24(6):510–529, 2010.

[112] M. Herman and J. S. Albus. Real-time hierarchical planning for multiple mobile robots. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, pages 22–1 – 22–10. 1987.

[113] D. Heylen, S. Kopp, S. C. Marsella, C. Pelachaud, and H. H. Vilhjálmsson. The Next Step towards a Function Markup Language. In H. Prendinger, J. C. Lester, and M. Ishizuka, editors, *Proceedings of the 8th International Conference on Intelligent Virtual Agents*, volume 5208 of *Lecture Notes in Artificial Intelligence*, pages 270–280. Springer, 2008.

[114] J. K. Hodgins and N. S. Pollard. Adapting simulated behaviors for new characters. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 153–162. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

[115] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 71–78. ACM Press, New York, NY, USA, 1995.

[116] D. H. W. Hofs, M. Theune, and R. op den Akker. Natural interaction with a virtual guide in a virtual environment: A multimodal dialogue system. *Journal on Multimodal User Interfaces*, 3(1-2):141–153, March 2010.

[117] E. Hsu, K. Pulli, and J. Popović. Style translation for human motion. *ACM Transactions on Graphics*, 24(3):1082–1089, 2005.

[118] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson. Arrows, Robots, and Functional Reactive Programming. In J. Jeuring and S. Jones, editors, *Advanced Functional Programming*, volume 2638 of *Lecture Notes in Computer Science*, pages 1949–1949. Springer Berlin / Heidelberg, 2003.

[119] Humanoid Animation Working Group. H|Anim, 2005. Http://www.h-anim.org/.

[120] L. Ik Soo and D. Thalmann. Construction of animation models out of captured data. In *Proceedings IEEE International Conference on Multimedia and Expo*, pages 829 – 832. IEEE, 2002.

[121] L. Ikemoto, O. Arikan, and D. Forsyth. Quick transitions with cached multi-way blends. In *I3D '07: Proceedings of the symposium on Interactive 3D graphics and games*, pages 145–151. ACM, New York, NY, USA, 2007.

[122] L. K. M. Ikemoto, O. Arikan, and D. A. Forsyth. Knowing when to put your foot down. In *I3D '06: Proceedings of the symposium on Interactive 3D graphics and games*, pages 49–53. ACM, New York, NY, USA, 2006.

[123] P. M. Isaacs and M. F. Cohen. Controlling dynamic simulation with kinematic constraints. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 215–224. ACM, New York, NY, USA, 1987.

[124] W.-S. Jang, W.-K. Lee, I.-K. Lee, and J. Lee. Enriching a motion database by analogous combination of partial human motions. *The Visual Computer*, 24(4):271–280, 2008.

[125] S. E. M. Jansen and H. van Welbergen. Methodologies for the User Evaluation of the Motion of Virtual Humans. In Z. M. Ruttkay, M. Kipp, A. Nijholt, and H. H. Vilhjálmsson, editors, *Proceedings of the 9th International Conference on Intelligent Virtual Agents*, volume 5773 of *Lecture Notes in Computer Science*, pages 125–131. Springer Berlin / Heidelberg, Berlin, 2009.

[126] L. L. Johnson, J. W. Rickel, and J. Lester. Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *International Journal of Artificial Intelligence in Education*, 11:47–78, 2000.

[127] T. de Jong. Technological Advances in Inquiry Learning. *Science*, 28:532–533, April 2006.

[128] G. R. Jonsdottir, J. Gratch, E. Fast, and K. R. Thórisson. Fluid Semantic Back-Channel Feedback in Dialogue: Challenges and Progress. In C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis, and D. Pelé, editors, *Proceedings of the 7th International Conference on Intelligent Virtual Agents*, volume 4722 of *Lecture Notes in Computer Science*, pages 154–160. Springer Berlin / Heidelberg, 2007.

[129] G. R. Jonsdottir and K. R. Thórisson. Teaching Computers to Conduct Spoken Interviews: Breaking the Realtime Barrier with Learning. In Z. M. Ruttkay, M. Kipp, A. Nijholt, and H. H. Vilhjálmsson, editors, *Proceedings of the 9th International Conference on Intelligent Virtual Agents*, volume 5773 of *Lecture Notes in Computer Science*, pages 446–459. 2009.

[130] G. R. Jonsdottir, K. R. Thórisson, and E. Nivel. Learning Smooth, Human-Like Turntaking in Realtime Dialogue. In H. Prendinger, J. C. Lester, and M. Ishizuka, editors, *Proceedings of the 8th International Conference on Intelligent Virtual Agents*, volume 5208 of *Lecture Notes in Artificial Intelligence*, pages 162–175. Springer-Verlag, 2008.

[131] M. Kallmann, S. C. Marsella, D. Ballin, and P. Olivier. Hierarchical motion controllers for real-time autonomous virtual humans. In T. Panayiotopoulos, J. Gratch, R. S. Aylett, P. Olivier, and T. Rist, editors, *Proceedings of the 5th International Conference on Intelligent Virtual Agents*, volume 3661 of *Lecture Notes in Computer Science*, pages 253–265. Springer-Verlag, London, UK, 2005.

[132] M. Kawato. Internal Models for Motor Control and Trajectory Planning. *Current Opinion in Neurobiology*, 9:718–727, 1999.

[133] J. A. S. Kelso. Phase transitions and critical behavior in human bimanual coordination. *American Journal of Physiology*, 246(6):R1000–R1004, 1984.

[134] J. A. S. Kelso. *Dynamic Patterns: The Self-Organization of Brain and Behavior*. MIT Press, Cambridge, MA, USA, 1995.

[135] J. A. S. Kelso, J. Buchanan, and W. S. A. Order parameters for the neural organization of single, multijoint limb movement patterns. *Experimental Brain Research*, 85(2):432–444, 1991.

[136] J. A. S. Kelso and J. J. Jeka. Symmetry breaking dynamics of human multilimb coordination. *Journal of Experimental Psychology, Human Perception and Performance*, 18(3):645–668, 1992.

[137] J. A. S. Kelso and S. D. L. On the nature of human interlimb coordination. *Science*, 203(4384):1029–1031, 1979.

[138] A. Kendon. Movement coordination in social interaction: Some examples described. *Acta Psychologica*, 32:101–125, 1970.

[139] A. Kendon. Gesticulation and Speech: Two Aspects of the Process of Utterance. In M. R. Key, editor, *The relation of verbal and nonverbal communication*, pages 207–227. Mouton, The Hague, The Netherlands, 1980.

[140] A. Kendon. An Agenda for Gesture Studies. *The Semiotic Review of Books*, 7(3):7–12, 1996.

[141] P. Kenny, A. Hartholt, J. Gratch, W. Swartout, D. Traum, S. C. Marsella, and D. Piepol. Building Interactive Virtual Humans for Training Environments. In *Interservice/Industry Training, Simulation, and Education Conference*, pages 1–16. 2007.

[142] M. S. van Kessel. *Gestures of a virtual guide*. Master's thesis, University of Twente, 2006.

[143] M.-J. Kim, M.-S. Kim, and S. Y. Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 369–376. ACM, New York, NY, USA, 1995.

[144] M. Kipp, A. Heloir, M. Schröder, and P. Gebhard. Realizing Multimodal Behavior: Closing the gap between behavior planning and embodied agent presentation. In J. Allbeck, N. Badler, T. W. Bickmore, C. Pelachaud, and A. Safonova, editors, *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, volume 6356 of *Lecture Notes in Computer Science*, pages 57–63. Springer, 2010.

[145] H.-S. Ko and N. I. Badler. Animating Human Locomotion with Inverse Dynamics. *IEEE Computer Graphics and Applications*, 16(2):50–59, 1996.

[146] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 33–41. ACM Press, New York, NY, USA, 1984.

[147] I. de Kok and D. Heylen. Multimodal end-of-turn prediction in multi-party meetings. In *ICMI-MLMI '09: Proceedings of the international conference on Multimodal interfaces*, pages 91–98. ACM, New York, NY, USA, 2009.

[148] I. de Kok and D. Heylen. Appropiate and Inappropiate Timing of Listener Responses from Multiple Perspectives. In *Proceedings of the 11th International Conference on Intelligent Virtual Agents*. 2011. To appear.

[149] E. Kokkevis, D. N. Metaxas, and N. I. Badler. User-Controlled Physics-Based Animation for Articulated Figures. In *Proceedings of Computer Animation*, pages 16 – 26. IEEE Computer Society, Washington, DC, USA, 1996.

[150] S. Kopp. Surface Realization of Multimodal Output from XML representations in MURML. In *Workshop on Representations for Multimodal Generation*. April 2005.

[151] S. Kopp. Social resonance and embodied coordination in face-to-face conversation with artificial interlocutors. *Speech Communication*, 52(6):587–597, 2010.

[152] S. Kopp, B. Krenn, S. C. Marsella, A. N. Marshall, C. Pelachaud, H. Pirker, K. R. Thórisson, and H. H. Vilhjálmsson. Towards a Common Framework for Multimodal Generation: The Behavior Markup Language. In J. Gratch, M. R. Young, R. S. Aylett, D. Ballin, and P. Olivier, editors, *Proceedings of the 6th International Conference on Intelligent Virtual Agents*, volume 4133 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2006.

[153] S. Kopp, T. Stocksmeier, and D. Gibbon. Incremental Multimodal Feedback for Conversational Agents. In C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis, and D. Pelé, editors, *Proceedings of the 7th International Conference on Intelligent Virtual Agents*, volume 4722 of *Lecture Notes in Computer Science*, pages 139–146. Springer Berlin / Heidelberg, 2007.

[154] S. Kopp and I. Wachsmuth. Model-based Animation of Coverbal Gesture. In *Computer Animation*, pages 252–260. 2002.

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

[156] L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *SCA '03: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 214–224. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003.

[157] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. In *ACM Transactions on Graphics*, volume 23, pages 559–568. ACM Press, New York, NY, USA, 2004.

[158] L. Kovar, M. Gleicher, and F. H. Pighin. Motion graphs. In *ACM Transactions on Graphics*, volume 3 of *21*, pages 473–482. ACM, 2002.

[159] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *SCA '02: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–104. ACM Press, New York, NY, USA, 2002.

[160] R. M. Krauss, P. D. Bricker, L. E. McMahon, and C. M. Garlock. The role of audible and visible back channel responses in interpersonal communication. *Journal of Personality and Social Psychology*, 35(7):523–529, 1977.

[161] K. Kuiper. *Smooth talkers: The linguistic performance of auctioneers and sportscasters*. Everyday communication: Case studies of behavior in context. Lawrence Erlbaum Associates Inc, Mahwah, NJ, USA, 1995.

[162] D. Lakens. Movement synchrony and perceived entitativity. *Journal of Experimental Social Psychology*, 46(5):701–708, 2010.

[163] D. Lakens and M. Stel. If they move in sync, they must feel in sync: Movement synchrony leads to attributed feelings of rapport. *Social Cognition*, 29(1), 2010.

[164] J. L. Lakin, V. E. Jefferis, M. Cheng, and T. L. Chartrand. The Chameleon Effect as social glue: Evidence for the evolutionary significance of nonconscious mimicry. *Journal of Nonverbal Behavior*, 27(3):145–162, 2003.

[165] B. J. Lance and S. C. Marsella. A Model of Gaze for the Purpose of Emotional Expression in Virtual Embodied Agents. In *AAMAS '08: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 199–206. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[166] M. Lau, Z. Bar-Joseph, and J. Kuffner. Modeling spatial and temporal variation in motion data. *ACM Transactions on Graphics*, 28(5):171:1–171:10, December 2009.

[167] J. Lee. Representing Rotations and Orientations in Geometric Computing. *IEEE Computer Graphics and Applications*, 28(2):75 –83, march-april 2008.

[168] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive Control of Avatars Animated With Human Motion Data. *ACM Transactions on Graphics*, 21(3):491–500, July 2002.

[169] J. Lee and K. H. Lee. Precomputing avatar behavior from human motion data. *Graphical Models*, 68(2):158–174, 2006.

[170] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[171] P. Lee, S. Wei, J. Zhao, and N. I. Badler. Strength guided motion. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 253–262. ACM, New York, NY, USA, 1990.

[172] Y. Lee, S. J. Lee, and Z. Popović. Compact character controllers. *ACM Transactions on Graphics*, 28(5):1–8, 2009.

[173] Y. Li, T. Wang, and H.-Y. Shum. Motion texture: a two-level statistical model for character motion synthesis. In *ACM Transactions on Graphics*, volume 21, pages 465–472. ACM Press, New York, NY, USA, 2002.

[174] K. C. Liu and Z. Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics*, 21(3):408–416, 2002.

[175] W.-Y. Lo and M. Zwicker. Real-time planning for parameterized human motion. In *SCA '08: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 29–38. Eurographics Association, Aire-la-Ville, Switzerland, 2008.

[176] D. P. Loehr. *Gesture and Intonation*. Ph.D. thesis, Graduate School of Arts and Sciences of Georgetown University, 2004.

[177] A. B. Loyall, W. S. N. Reilly, J. Bates, and P. Weyhrauch. System for authoring highly interactive, personality-rich interactive characters. In *SCA '04: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 59–68. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2004.

[178] M. ter Maat, R. M. Ebbers, D. Reidsma, and A. Nijholt. Beyond the beat: modelling intentions in a virtual conductor. In *Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*, pages 12:1–12:10. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels, Belgium, 2007.

[179] M. ter Maat and D. Heylen. Turn Management or Impression Management? In Z. M. Ruttkay, M. Kipp, A. Nijholt, and H. H. Vilhjálmsson, editors, *Proceedings of the 9th International Conference on Intelligent Virtual Agents*, volume 5773 of *Lecture Notes in Computer Science*, pages 467–473. Springer Berlin / Heidelberg, 2009.

[180] M. ter Maat and D. Heylen. Flipper: An Information State Component for Spoken Dialogue Systems. In *Proceedings of the 11th International Conference on Intelligent Virtual Agents*. 2011. To appear.

[181] M. ter Maat, K. P. Truong, and D. Heylen. How turn-taking strategies influence users' impressions of an agent. In J. Allbeck, N. I. Badler, T. W. Bickmore, C. Pelachaud, and A. Safonova, editors, *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, volume 6356 of *Lecture Notes in Computer Science*, pages 441–453. Springer, Philadelphia, Pennsylvania, USA, 2010.

[182] R. Maatman, J. Gratch, and S. C. Marsella. Natural behavior of a listening agent. In T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, and T. Rist, editors, *Proceedings of the 5th International Conference on Intelligent Virtual Agents*, volume 3661 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2005.

[183] N. A. Macmillan and D. C. Creelman. *Detection Theory: A User's Guide*. Lawrence Erlbaum, New York, NY, USA, second edition, 2004.

[184] N. Magnenat-Thalmann, H. Seo, and F. Cordier. Automatic modeling of virtual humans and body clothing. *Journal of Computer Science and Technologie*, 19(5):575–584, 2004.

[185] N. Magnenat-Thalmann and D. Thalmann. Computer animation. *ACM Computing Surveys*, 28(1):161–163, 1996.

[186] M. Mancini, R. Niewiadomski, E. Bevacqua, and C. Pelachaud. Greta: a SAIBA compliant ECA system. In *Troisiéme Workshop sur les Agents Conversationnels Animés*. 2008.

[187] M. J. Mandel. *Versatile and Interactive Virtual Humans: Hybrid use of Data-Driven and Dynamics-Based Motion Synthesis*. Master's thesis, Carnegie Mellon University, August 2004.

[188] T. Mathworks. Matlab Optimization Toolbox. Http://www.mathworks.com/products/optimization/.

[189] D. McNeill. *Hand and Mind: What Gestures Reveal about Thought.* University of Chicago Press, Chicago, IL, USA, 1995.

[190] D. McNeill. Catchments and contexts: non-modular factors in speech and gesture production. In D. McNeill, editor, *Language and gesture*, chapter 15. Cambridge University Press, Cambridge, 2000.

[191] F. Mechsner, D. Kerzel, G. Knoblich, and W. Prinz. Perceptual basis of bimanual coordination. *Nature*, 414(6859):69–73, 2001.

[192] S. Ménardais, R. Kulpa, F. Multon, and B. Arnaldi. Synchronization for dynamic blending of motions. In *SCA '04: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 325–335. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2004.

[193] G. Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, Boston, MA, USA, 2007.

[194] L. K. Miles, L. K. Nind, and C. N. Macrae. The rhythm of rapport: Interpersonal synchrony and social perception. *Journal of Experimental Social Psychology*, 45(3):585 – 589, 2009.

[195] B. Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(2):31–50, 1996.

[196] M. Mizuguchi, J. Buchanan, and T. Calvert. Data driven motion transitions for interactive games. In *Short paper proceedings of Eurographics*. 2001.

[197] L.-P. Morency, I. de Kok, and J. Gratch. A probabilistic multimodal approach for predicting listener backchannels. *Autonomous Agents and Multi-Agent Systems*, 20(1):70–84, May 2010.

[198] U. Muico, Y. Lee, J. Popović, and Z. Popović. Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics*, 28(3):1–9, 2009.

[199] T. Mukai and S. Kuriyama. Geostatistical motion interpolation. *ACM Transactions on Graphics*, 24(3):1062–1070, 2005.

[200] NaturalMotion. Euphoria and Endorphin. Http://www.naturalmotion.com/.

[201] M. Neff. *Aesthetic Exploration and Refinement: A Computational Framework for Expressive Character Animation*. Ph.D. thesis, Department of Computer Science, University of Toronto, 2005.

[202] M. Neff and E. Fiume. Modeling tension and relaxation for computer animation. In *SCA '02: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 81–88. ACM Press, New York, NY, USA, 2002.

[203] M. Neff and E. Fiume. AER: aesthetic exploration and refinement for expressive character animation. In *SCA '05: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–170. ACM Press, New York, NY, USA, 2005.

[204] M. Neff and E. Fiume. Methods for exploring expressive stance. *Graphical Models*, 68(2):133–157, 2006.

[205] M. Neff and E. Fiume. From Performance Theory to Character Animation Tools. In *Human Motion - Understanding, Modelling, Capture, and Animation*, pages 597–629. Springer Netherlands, 2008.

[206] M. Neff and Y. Kim. Interactive editing of motion style using drives and correlations. In *SCA '09: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 103–112. ACM, New York, NY, USA, 2009.

[207] M. Neff, M. Kipp, I. Albrecht, and H.-P. Seidel. Gesture Modeling and Animation Based on a Probalistic Recreation of Speaker Style. *ACM Transactions on Graphics*, 27(1):1–24, 2008.

[208] D. Neiberg and K. P. Truong. Online Detection Of Vocal Listener Responses With Maximum Latency Constraints. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5836–2539. September 2011.

[209] H. Noot and Z. M. Ruttkay. Variations in gesturing and speech by GESTYLE. *International Journal of Human-Computer Studies*, 62(2):211 – 229, 2005.

[210] Nvidia. Nvidia PhysX, 2008. Http://www.nvidia.com/.

[211] S. Oore, D. Terzopoulos, and G. E. Hinton. Local Physical Models for Interactive Character Animation. *Computer Graphics Forum*, 21(3):337–346, September 2002.

[212] M. Oshita. Smart Motion Synthesis. *Computer Graphics Forum*, 27(7):1909–1918, 2008.

[213] E. Otten. Inverse and forward dynamics: models of multi-body systems. *Philosophical Transactions of the Royal Society*, 358(1437):1493–1500, September 2003.

[214] S. Pammi. Synthesis of nonverbal listener vocalizations. In *Doctorial consortium at International Conference on Affective Computing & Intelligent Interaction*. 2009.

[215] I. S. Pandžić and R. Forchheimer. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. Wiley, 2002.

[216] R. Paul. *Realization and high level specification of facial expressions for embodied agents*. Master's thesis, University of Twente, 2010.

[217] K. Perlin. Real Time Responsive Animation with Personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.

[218] K. Perlin and A. Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 205–216. ACM, New York, NY, USA, 1996.

[219] M. J. Pickering and S. Garrod. Toward a mechanistic psychology of dialogue. *Behavioral and Brain Sciences*, 27(2):169–190, 2004.

[220] P. Piwek, B. Krenn, M. Schröder, M. Grice, S. Baumann, and H. Pirker. RRL: A Rich Representation Language for the Description of Agent Behaviour in NECA. In *Proceedings of the "Embodied conversational agents - let's specify and evaluated them!" workshop at AAMAS'02*. 2004.

[221] R. Plutchik. *Emotion: A Psychoevolutionary Synthesis*. Harper and Row, New York, NY, USA, 1980.

[222] N. S. Pollard and P. S. A. Reitsma. Animation of humanlike characters: Dynamic motion filtering with a physically plausible contact model. In *Yale Workshop on Adaptive and Learning Systems*. 2001.

[223] R. Poppe, K. P. Truong, and D. Heylen. Backchannels: Quantity, Type and Timing Matters. In *Proceedings of the 11th International Conference on Intelligent Virtual Agents*. 2011. To appear.

[224] R. Poppe, K. P. Truong, D. Reidsma, and D. Heylen. Backchannel Strategies for Artificial Listeners. In J. Allbeck, N. Badler, T. W. Bickmore, C. Pelachaud, and A. Safonova, editors, *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, volume 6356 of *Lecture Notes in Computer Science*, pages 146–158. Springer Berlin / Heidelberg, 2010.

[225] P. Radua, D. Tweed, and T. Vilis. Three-dimensional eye, head, and chest orientations after large gaze shifts and the underlying neural strategies. *Journal of Neurophysiology*, 72(6):2840–2852, 1994.

[226] A. Raouzaiou, N. Tsapatsoulis, K. Karpouzis, and S. Kollias. Parameterized facial expression synthesis based on MPEG-4. *Journal of Applied Signal Processing*, 2002(1):1021–1038, 2002.

[227] B. Reeves and C. Nass. *The media equation: how people treat computers, television, and new media like real people and places*. Cambridge University Press, New York, NY, USA, 1996.

[228] D. Reidsma, I. de Kok, D. Neiberg, S. Pammi, B. van Straalen, K. P. Truong, and H. van Welbergen. Continuous Interaction with a Virtual Human. *Journal on Multimodal User Interfaces*, 4(2):97–118, 2011.

[229] D. Reidsma, A. Nijholt, and P. Bos. Temporal interaction between an artificial orchestra conductor and human musicians. *Computers in Entertainment*, 6(4):1–22, 2008.

[230] D. Reidsma, H. van Welbergen, R. Poppe, P. Bos, and A. Nijholt. Towards Bi-directional Dancing Interaction. In R. Harper, M. Rauterberg, and M. Combetto, editors, *ICEC 2006: Proceedings of the 5th International Conference on Entertainment Computing*, volume 4161 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2006.

[231] P. S. A. Reitsma, J. Andrews, and N. S. Pollard. Effect of Character Animacy and Preparatory Motion on Perceptual Magnitude of Errors in Ballistic Motion. *Computer Graphics Forum*, 27(2):201–210, 2008.

[232] P. S. A. Reitsma and N. S. Pollard. Perceptual Metrics for Character Animation: Sensitivity to Errors in Ballistic Motion. *ACM Transactions on Graphics*, 22(3):537–542, July 2003.

[233] P. S. A. Reitsma and N. S. Pollard. Evaluating Motion Graphs for Character Animation. *ACM Transactions on Graphics*, 26(4):18, October 2007.

[234] L. Ren, A. Patrick, A. A. Efros, J. K. Hodgins, and J. M. Rehg. A Data-Driven Approach to Quantifying Natural Human Motion. *ACM Transactions on Graphics*, 24(3):1090–1097, August 2005.

[235] J. Rickel and W. L. Johnson. STEVE: A Pedagogical Agent for Virtual Reality. In K. P. Sycara and M. Wooldridge, editors, *Agents'98: Proceedings of the 2nd International Conference on Autonomous Agents*, pages 332–333. ACM Press, New York, 1998.

[236] J. W. Rickel, J. Gratch, S. C. Marsella, and W. Swartout. Steve goes to Bosnia: Towards a new generation of virtual humans for interactive experiences. In *AAAI Spring Symposium of Artificial Intelligence and Interactive Entertainment*. 2001.

[237] A. Ridderikhoff, C. L. E. Peper, and P. J. Beek. Unraveling Interlimb Interactions Underlying Bimanual Coordination. *Journal of Neurophysiology*, 94:3112–3125, 2005.

[238] J. Riskin. *Genesis Redux*. The University of Chicago Press, Chicago, IL, USA, 2007.

[239] L. F. Robinson and H. T. Reis. The effects of interruption, gender, and status on interpersonal perceptions. *Journal of Nonverbal Behavior*, 13:141–153, 1989.

[240] C. F. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[241] C. F. Rose, B. Guenter, B. Bodenheimer, and M. F. Cohen. Efficient generation of motion transitions using space-time constraints. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 147–154. ACM Press, New York, NY, USA, 1996.

[242] C. F. Rose, P.-P. J. Sloan, and M. F. Cohen. Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation. *Computer Graphics Forum*, 20:239–250(12), September 2001.

[243] J. P. de Ruiter. The production of gesture and speech. In D. McNeill, editor, *Language and gesture*, chapter 14. Cambridge University Press, Cambridge, 2000.

[244] J. P. de Ruiter, H. Mitterer, and N. J. Enfield. Projecting the end of a speaker's turn: A cognitive cornerstone of conversation. *Language*, 82(3):515–535, 2006.

[245] K. B. Russell and B. M. Blumberg. Behavior-Friendly Graphics. In *Proceedings of Computer Graphics International*, pages 44–50. IEEE Computer Society, Washington, DC, USA, 1999.

[246] Z. M. Ruttkay and C. Pelachaud. Excercises Of Style For Virtual Humans. In *Proceedings of Animating Expressive Characters for Social Interaction Symposium*, pages 85–90. Imperial College, 2002.

[247] Z. M. Ruttkay and H. van Welbergen. Elbows higher! Performing, observing and correcting exercises by a Virtual Trainer. In H. Prendinger, J. C. Lester, and M. Ishizuka, editors, *Proceedings of the 8th International Conference on Intelligent Virtual Agents*, volume 5208 of *Lecture Notes in Artificial Intelligence*, pages 409–416. Springer-Verlag, Berlin, Heidelberg, 2008.

[248] Z. M. Ruttkay, J. Zwiers, H. van Welbergen, and D. Reidsma. Towards a Reactive Virtual Trainer. In J. Gratch, M. Young, R. S. Aylett, D. Ballin, and P. Olivier, editors, *Proceedings of the 6th International Conference on Intelligent Virtual Agents*, volume 4133 of *Lecture notes in Computer Science*, pages 292–303. Springer Verlag, Berlin, 2006.

[249] H. Sacks, E. Schegloff, and G. Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50:696–735, 1974.

[250] A. Safonova and J. K. Hodgins. Analyzing the Physical Correctness of Interpolated Human Motion. In *SCA '05: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 171–180. ACM, New York, NY, USA, August 2005.

[251] E. Schegloff. Overlapping talk and the organization of turn-taking for conversation. *Language in Society*, 29:1–63, 2000.

[252] D. Schlangen. From Reaction To Prediction: Experiments with Computational Models of Turn-Taking. In *Proceedings of Interspeech*. 2006.

[253] D. Schlangen and G. Skantze. A general, abstract model of incremental dialogue processing. In *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 710–718. Association for Computational Linguistics, Stroudsburg, PA, USA, 2009.

[254] R. A. Schmidt. A schema theory of discrete motor skill learning. *Psychological Review*, 82(4):225–260, 1975.

[255] R. C. Schmidt, C. Carello, and M. T. Turvey. Phase transitions and critical fluctuations in the visual coordination of rhythmic movements between people. *Journal of Experimental Psychology, Human Perception and Performance*, 16(2):227–247, 1990.

[256] R. C. Schmidt and M. Richardson. Dynamics of Interpersonal Coordination. In A. Fuchs and V. Jirsa, editors, *Coordination: Neural, Behavioral and Social Dynamics*, volume 17 of *Understanding Complex Systems*, pages 281–308. Springer Berlin / Heidelberg, 2008.

[257] M. Schröder. The SEMAINE API: Towards a Standards-Based Framework for Building Emotion-Oriented Systems. *Advances in Human-Computer Interaction*, 2010(319406), 2010.

[258] M. Schröder, R. Cowie, D. K. J. Heylen, M. Pantic, C. Pelachaud, and B. Schuller. Towards responsive Sensitive Artificial Listeners. In *Proceedings of the 4th International Workshop on Human-Computer Conversation*. University of Sheffield, Sheffield, UK, October 2008.

[259] A. Shapiro, Y. Cao, and P. Faloutsos. Style components. In *Proceedings of Graphics Interface*, pages 33–39. Canadian Information Processing Society, Toronto, Ont., Canada, 2006.

[260] A. Shapiro, P. Faloutsos, and V. Ng-Thow-Hing. Dynamic Animation and Control Environment. In *Proceedings of Graphics Interface*, pages 61–70. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005.

[261] A. Shapiro, M. Kallmann, and P. Faloutsos. Interactive motion correction and object manipulation. In *I3D '07: Proceedings of the symposium on Interactive 3D graphics and games*, pages 137–144. ACM, New York, NY, USA, 2007.

[262] A. Shapiro, F. H. Pighin, and P. Faloutsos. Hybrid Control for Interactive Character Animation. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 455–461. IEEE Computer Society, Washington, DC, USA, 2003.

[263] D. Sharon and M. van de Panne. Synthesis of Controllers for Stylized Planar Bipedal Walking. In *Proceedings of the IEEE International Conference on Robotics and Animation*, pages 2387–2392. 2005.

[264] M. Sherman and D. Rosenthal. SD/FAST, 2001. Http://www.sdfast.com/.

[265] H. J. Shin, L. Kovar, and M. Gleicher. Physical Touch-Up of Human Motions. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 194–203. IEEE Computer Society, Washington, DC, USA, 2003.

[266] H. J. Shin and H. S. Oh. Fat graphs: constructing an interactive character with continuous controls. In *SCA '06: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 291–298. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2006.

[267] H. P. H. Shum, T. Komura, and S. Yamazaki. Simulating interactions of avatars in high dimensional state space. In *I3D '08: Proceedings of the symposium on Interactive 3D graphics and games*, pages 131–138. ACM, New York, NY, USA, 2008.

[268] M. da Silva, Y. Abe, and J. Popović. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics*, 27(3):82, August 2008.

[269] M. da Silva, Y. Abe, and J. Popović. Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Computer Graphics Forum*, 27(2):371–380, April 2008.

[270] K. Sims. Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, New York, NY, USA, 1994.

[271] G. Skantze and A. Hjalmarsson. Towards Incremental Speech Generation in Dialogue Systems. In *SIGDIAL '10: Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 1–8. Association for Computational Linguistics, Stroudsburg, PA, USA, 2010.

[272] M. Slee, A. Agarwal, and M. Kwiatkowski. Thrift: Scalable Cross-Language Services Implementation, 2007.

[273] R. Smith. Open Dynamics Engine, 2008. Http://www.ode.org/.

[274] S. B. Software. Optimization Software. Http://www.sbsi-sol-optimize.com/index.htm.

[275] G. Solano. A BMLbased Embodied Conversational Agent for a Personality Detection Program. In *Proceedings of the 11th International Conference on Intelligent Virtual Agents*. 2011. To appear.

[276] D. Sternad, W. J. Dean, and S. Schaal. Interaction of rhythmic and discrete pattern generators in single-joint movements. *Human Movement Science*, 19(4):627 – 664, 2000.

[277] A. J. Stewart and J. F. Cremer. Beyond keyframing: an algorithmic approach to animation. In *Proceedings of Graphics Interface*, pages 273–281. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[278] D. Thalmann. Motion Modeling: Can We Get Rid of Motion Capture? In A. Egges, A. Kamphuis, and M. Overmars, editors, *Motion in Games*, volume 5277 of *Lecture Notes in Computer Science*, pages 121–131. Springer-Verlag, 2008.

[279] M. Theune, D. Heylen, and A. Nijholt. Generating Embodied Information Presentations. In O. Stock and Zancaro, editors, *Multimodal Intelligent Information Presentation*, pages 47–69. Kluwer Academic Publishers, 2005.

[280] M. Thiebaux, A. N. Marshall, S. C. Marsella, and M. Kallmann. SmartBody: Behavior Realization for Embodied Conversational Agents. In *AAMAS '08: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 151–158. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[281] B. Thór Árnason and Æ. Thorsteinsson. The CADIA BML Realizer, 2008. Http://cadia.ru.is/projects/bmlr/.

[282] K. R. Thórisson. *Communicative Humanoids: A computational Model of Psychosocial Dialog Skills*. Ph.D. thesis, Massachusetts Institute of Technology, 1996.

[283] K. R. Thórisson. A Mind Model for Multimodal Communicative Creatures & Humanoids. *Applied Artificial Intelligence*, 13(4-5):449–486, 1999.

[284] K. R. Thórisson. Natural Turn-Taking Needs No Manual: Computational Theory and Model, from Perception to Action. In B. Granström, D. House, and I. Karlsson, editors, *Multimodality in Language and Speech Systems*, pages 173–207. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[285] K. R. Thórisson, H. Benko, D. Abramov, A. Arnold, S. Maskey, and A. Vaseekaran. Constructionist Design Methodology for Interactive Intelligences. *AI Magazine*, 25(4), 2004.

[286] K. R. Thórisson and H. H. Vilhjálmsson. Functional Description of Multimodal Acts: A Proposal. In *Proceedings of the 2nd Function Markup Language Workshop at AAMAS '09*. 2009.

[287] D. Tolani, A. Goswami, and N. I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models and Image Processing*, 62(5):353–388, 2000.

[288] L. Torresani, P. Hackney, and C. Bregler. Learning Motion Style Synthesis from Perceptual Observations. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19, Proceedings of the 20th Annual Conference on Neural Information Processing Systems*, pages 1393–1400. MIT Press, Cambridge, MA, 2007.

[289] P. Touset. *Natural Language Processing for a Psychometer: A dialogue system for the measurement of personality traits*. Master's thesis, University of Twente, 2007.

[290] P. Treffner and M. Peter. Intentional and attentional dynamics of speech-hand coordination. *Human Movement Science*, 21(5-6):641 – 697, 2002.

[291] P. J. Treffner and M. T. Turvey. Symmetry, broken symmetry, and handedness in bimanual coordination dynamics. *Experimental Brain Research*, 107(3):463–478, January 1996.

[292] A. Treuille, Y. Lee, and Z. Popović. Near-optimal character animation with continuous control. *ACM Transactions on Graphics*, 26(3):7, 2007.

[293] D. Tweed. Three-dimensional model of the human eye-head saccadic system. *Journal of Neurophysiology*, 77(2):654–666, February 1997.

[294] D. Tweed and T. Vilis. Listing's Law for Gaze-Directing Head Movements. In A. Berthoz, W. Graf, and P. P. Vidal, editors, *The Head-Neck Sensory-Motor System*, pages 387–391. Oxford University Press, New York, 1992.

[295] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement - Minimum torque-change model. *Biological Cybernetics*, 61:89–101, 1989.

[296] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 91–96. ACM Press, New York, NY, USA, 1995.

[297] R. Urtasun, P. Glardon, R. Boulic, D. Thalmann, and P. Fua. Style-based motion synthesis. *Computer Graphics Forum*, 23(4):799–812, 2004.

[298] H. H. Vilhjálmsson, N. Cantelmo, J. Cassell, N. E. Chafai, M. Kipp, S. Kopp, M. Mancini, S. C. Marsella, A. N. Marshall, C. Pelachaud, Z. M. Ruttkay, K. R. Thórisson, H. van Welbergen, and R. J. van der Werf. The Behavior Markup Language: Recent Developments and Challenges. In C. Pelachaud, J. C. Martin, E. Andre, G. Collet, K. Karpouzis, and D. Pelé, editors, *Proceedings of the 7th International Conference on Intelligent Virtual Agents*, volume 4722 of *Lecture Notes in Computer Science*, pages 99–120. Springer, Berlin, 2007.

[299] P. Viviani and C. Terzuolo. Trajectory determines movement dynamics. *Neuroscience*, 7(2):431–437, 1982.

[300] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel. Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Technical Report SMLI TR2004-0811, Sun Microsystems Inc., 2004.

[301] N. Ward and W. Tsukahara. Prosodic features which cue back-channel responses in English and Japanese. *Journal of Pragmatics*, 32(8):1177–1207, 2000.

[302] L. Weber, S. W. Smoliar, and N. I. Badler. An architecture for the simulation of human movement. In *ACM '78: Proceedings of the 1978 annual ACM conference - Volume 2*, pages 737–745. ACM, New York, NY, USA, 1978.

[303] H. van Welbergen. *A virtual human presenter in a 3D meeting environment*. Master's thesis, University of Twente, Enschede, the Netherlands, August 2005.

[304] H. van Welbergen, A. Nijholt, D. Reidsma, and J. Zwiers. Presenting in Virtual Worlds: Towards an Architecture for a 3D Presenter explaining 2D-Presented Information. *IEEE Intelligent Systems*, 21(5):47–53, 2006.

[305] D. J. Wiley and J. K. Hahn. Interpolation Synthesis of Articulated Figure Motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, 1997.

[306] K. D. Willmert. Visualizing Human Body Motion Simulations. *IEEE Computer Graphics and Applications*, 2(9):35–38, November 1982.

[307] D. A. Winter. *Biomechanics and Motor Control of Human Movement*. Wiley, Hoboken, NJ, USA, third edition, 2004.

[308] A. Witkin and M. Kass. Spacetime constraints. *SIGGRAPH Computer Graphics*, 22(4):159–168, 1988.

[309] A. Witkin and Z. Popović. Motion warping. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 105–108. ACM, New York, NY, USA, 1995.

[310] W. E. Woodson, B. Tillman, and P. Tillman. *Human Factors Design Handbook*. McGraw-Hill, second edition, 1992.

[311] W. L. Wooten and J. K. Hodgins. Transitions Between Dynamically Simulated Motions: Leaping, Tumbling, Landing, and Balancing. In *ACM SIGGRAPH Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97*, page 217. ACM, New York, NY, USA, 1997.

[312] W. L. Wooten and J. K. Hodgins. Simulating Leaping, Tumbling, Landing, and Balancing Humans. In *International Conference on Robotics and Animation*, pages 656–662. 2000.

[313] P. Wrotek, O. C. Jenkins, and M. McGuire. Dynamo: Dynamic, data-driven character control with adjustable balance. In *Sandbox '06: Proceedings of the ACM SIGGRAPH symposium on Videogames*, pages 61–70. ACM, New York, NY, USA, July 2006.

[314] Y. H. Yee and A. Newman. A perceptual metric for production testing. In *ACM SIGGRAPH Sketches*, page 121. ACM, New York, NY, USA, 2004.

[315] K. Yin, M. B. Cline, and D. K. Pai. Motion Perturbation Based on Simple Neuromotor Control Models. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 445–449. IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[316] K. Yin, S. Coros, P. Beaudoin, and M. van de Panne. Continuation Methods for Adapting Simulated Skills. *ACM Transactions on Graphics*, 27(3):1–7, 2008.

[317] K. Yin, K. Loken, and M. van de Panne. SIMBICON: simple biped locomotion control. *ACM Transactions on Graphics*, 26(3):105, 2007.

[318] V. H. Yngve. On getting a word in edgewise. *Papers from the Sixth Regional Meeting of the Chicago Linguistic Society*, pages 567–577, 1970.

[319] D. Zeltzer. Motor Control Techniques for Figure Animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, September 1982.

[320] X. Zhang and D. Chaffin. The effects of speed variation on joint kinematics during multi-segment reaching movements. *Human Movement Science*, 18:741–757, 1999.

[321] Y. Zhang, J. Qiang, Z. Zhu, and B. Yi. Dynamic Facial Expression Analysis and Synthesis With MPEG-4 Facial Animation Parameters. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(10):1383 – 1396, October 2008.

[322] L. Zhao and A. Safonova. Achieving Good Connectivity in Motion Graphs. In *SCA '08: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 127–136. Eurographics Association, Aire-la-Ville, Switzerland, 2008.

[323] V. B. Zordan and J. K. Hodgins. Tracking and Modifying Upper-Body Human Motion Data with Dynamic Simulation. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, pages 13–22. September 1999.

[324] V. B. Zordan and J. K. Hodgins. Motion capture-driven simulations that hit and react. In *SCA '02: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96. ACM, New York, NY, USA, 2002.

[325] V. B. Zordan, A. Macchietto, J. Medina, M. Soriano, and C.-C. Wu. Interactive dynamic response for games. In *Sandbox '07: Proceedings of the ACM SIGGRAPH symposium on Video games*, pages 9–14. ACM Press, New York, NY, USA, 2007.

[326] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. *ACM Transactions on Graphics*, 24(3):697–701, 2005.

Appendix A

Kinematics and Physics

Kinematic technologies can be used to control or analyze information of a kinematic nature, such as joint angles, joint angle velocity or joint angle acceleration. *Forward Kinematics* (FK) is the process of determining the position and/or rotation of an *end effector* (the joint at the end of a chain of joints) $\mathbf{s}$ given the rotations and translations $\mathbf{q}$ of all joints in the chain. *Inverse Kinematics* (IK) specifies the inverse problem: finding $\mathbf{q}$, given $\mathbf{s}$. Often this problem of finding joint rotations is underspecified, that is, there are multiple combinations of joint DoF values $\mathbf{q}$ that put the end effector in the right location. Several techniques exist to solve this problem. The IKAN toolkit [287] finds all joint configurations that solve the IK problem for an arm or leg. For larger chains numerical solutions are necessary. If these numerical techniques start out in a natural pose in which the end effector is already close to the goal, a natural pose will often be achieved. Boulic and Kulpa [37] provide an overview of commonly used numerical methods that solve the IK-problem and discuss the trade-offs made in naturalness and calculations speed in these different methods.

Kinematic based systems are intuitive, but do not explicitly model physical integrity. As a result, kinematic animation does not always seem to respond to gravity or inertia. Physical simulation models the body of the virtual human as a system of rigid bodies, connected by joints. Each of these rigid bodies has its own mass, inertia and possibly other physical properties. *Forward dynamics* is the animation process that moves a virtual human when the torques on its joints and its contact forces and impulses with the environment (through friction and collision) are provided. *Inverse dynamics* (ID) is the process of finding the torques and forces on the joints in a body given the movement of its segments. It can be used to predict torques needed for kinematically specified movement and to check if joint torques exceed strength limits. Several software toolkits can be used for forward dynamics and/or impact and friction handling. Boeing and Bräunl [35] provide a recent comparison of physics engines. Their benchmark software is available online and kept up to date with the latest physics engines. For real-time virtual human simulation, the accuracy and stability of the constraint satisfaction and the calculation time is important, but depending on the application the virtual human is used in, other simulation aspects, such as the accuracy of collision detection and friction handling could also play an

important role.

Appendix B

Conversion of Featherstone's 6D-vectors to traditional 3D vectors

Spatial velocity $\hat{\mathbf{v}}$ from velocity $\mathbf{v}$ and angular velocity ω:

$$\hat{\mathbf{v}} = \begin{bmatrix} \omega \\ \mathbf{v} \end{bmatrix} \tag{B.1}$$

Spatial acceleration $\hat{\mathbf{a}}$:

$$\hat{\mathbf{a}} = \begin{bmatrix} \dot{\omega} \\ \dot{\mathbf{v}} - \omega \times \mathbf{v} \end{bmatrix} \tag{B.2}$$

Spatial force $\hat{\mathbf{f}}$ from torque $\mathbf{n}$ and force $\mathbf{f}$:

$$\hat{\mathbf{f}} = \begin{bmatrix} \mathbf{n} \\ \mathbf{f} \end{bmatrix} \tag{B.3}$$

The spatial inertia tensor $\hat{\mathbf{I}}$ of a rigid body is a 6×6 vector, constructed from the inertial tensor at the CoM $\mathbf{I}^{cm}$, the mass m and the offset $\mathbf{c}$ of the CoM from the bodies origin.

$$\hat{\mathbf{I}} = \begin{bmatrix} \mathbf{I}^{cm} - m\mathbf{c} \times \mathbf{c}\times & m\mathbf{c}\times \\ -m\mathbf{c} & m\mathbf{1} \end{bmatrix} \qquad \mathbf{c}\times = \begin{bmatrix} 0 & -c_z & c_y \\ c_z & 0 & -c_x \\ -c_y & c_x & 0 \end{bmatrix} \tag{B.4}$$

Spatial cross product of force and velocity:

$$\hat{\mathbf{v}} \times^* \hat{\mathbf{f}} = \begin{bmatrix} \omega \times \mathbf{n} + \mathbf{v} \times \mathbf{f} \\ \omega \times \mathbf{f} \end{bmatrix} \tag{B.5}$$

Appendix C

The BML^T Specification

The BML Twente (BML^T) extension adds the specification of physical controllers, procedural animation, transition animations and continuous interaction capabilities to BML. Here I present an overview of the BMLT tags and their syntax. The namespace for BMLT is `http://hmi.ewi.utwente.nl/bmlt`, I use `bmlt` as the prefix for that namespace throughout this appendix. The most recent version of the BML^T specification can be found at `http://elckerlyc.ewi.utwente.nl/wiki/BMLT`.

C.1 The BML^T Elements

C.1.1 `<bmlt:parameter>`

Used to pass parameter values to other BMLT elements, such as `<bmlt:controller>`, `<bmlt:procanimation>`, `<bmlt:keyframe>` or `<bmlt:transition>`.

Attribute	Type	Use	Description
name	string	required	parameter name
value	string	required	parameter value

Table C.1: Attributes of the `<bmlt:parameter>` element.

C.1.2 `<bmlt:keyframe>`

Describes a keyframe (or mocap) animation. Parameter values can be set using `<bmlt:parameter>`. The joints parameter selects the joints the animation should act upon. This is useful to subtract motion on a subsection of the body from full body mocap. The mirror parameter (default false) mirrors the animation in the mid-sagittal plane. Mirror is always applied before joint selection.

Attribute	Type	Use	Description
name	string	required	name of the keyframe animation

Table C.2: Attributes of the `<bmlt:keyframe>` element.

BML Example 36 An example of the use of a `bmlt:keyframe` behavior.

```
<bmlt:keyframe id="v1" name="plain_rhand">
  <bmlt:parameter name="joints" value="r_shoulder r_elbow r_wrist"/>
  <bmlt:parameter name="mirror" value="false"/>
</bmlt:keyframe>
```

C.1.3 <bmlt:audiofile>

Behavior that plays an audio file. Useful to, for example, play a sound effect when the virtual human claps his hands.

Attribute	Type	Use	Description
filename	string	required	name of the file containing the audio

Table C.3: Attributes of the `<bmlt:audiofile>` element.

BML Example 37 An example of the use of a `bmlt:audiofile` behavior.

```
<bmlt:audiofile id="v1" filename="audio/clap.wav" start="3"/>
```

C.1.4 <bmlt:controller>

Describes a physical controller behavior. Parameters can be passed to the controller using `<bmlt:parameter>`.

Attribute	Type	Use	Description
class	string	required	Java class name of the controller

Table C.4: Attributes of the `<bmlt:controller>` element.

BML Example 38 Enable the balance controller with pelvis height at 1.3m (resulting in a balance motion with unbend legs) and slightly increased stiffness.

```
<bmlt:controller id="balance1" class="BalanceController" start="0" end="10">
  <bmlt:parameter name="pelvisheight" value="1.3"/>
  <bmlt:parameter name="stiffnessmultiplier" value="1.1"/>
</bmlt:controller>
```

C.1.5 <bmlt:procanimation>

Describes a procedural animations. Parameters can be passed using the BML^T element <bmlt:parameter>. Like in keyframe animation, the joints parameter selects the joints the animation should act upon. The mirror parameter (default false) mirrors the animation in the mid-sagittal plane. Again, mirror is always applied before joint selection.

Attribute	Type	Use	Description
name	string	required	name of the procedural animation

Table C.5: Attributes of the <bmlt:procanimation> element.

BML Example 39 Conduct a procedural 3 beat gesture with amplitude 5.

```
<bmlt:procanimation id="beat1" name="3-beat" start="3">
  <bmlt:parameter name="a" value="5"/>
</bmlt:procanimation>
```

C.1.6 <bmlt:transition>

Creates a transition animation. The start and end pose of the transition are determined automatically from its surrounding motions.

Attribute	Type	Use	Description
class	string	required	Java class name of the transition used

Table C.6: Attributes of the <bmlt:transition> element.

BML Example 40 Create a simple slerp transition on the arm's rotations, moving from the neutral pose toward the start arm pose of the conducting gesture.

```
<bmlt:transition id="trans1" class="SlerpTransition" start="1" end="2">
  <bmlt:parameter name="joints" value="r_shoulder,r_elbow,r_wrist,
    l_shoulder,l_elbow,l_wrist"/>
</bmlt:transition>
<bmlt:procanimation id="beat1" name="3-beat" start="trans1:end"/>
```

C.1.7 <bmlt:noise>

Generates a noise (typically Perlin noise) animation on a joint, specified by the type of noise, joint id and custom parameters.

Attribute	Type	Use	Description
joint	string	required	joint id
type	string	required	type of the noise

Table C.7: Attributes of the `<bmlt:noise>` element.

BML Example 41 Example of `bmlt:noise`.

```
<bmlt:noise id="noise1" type="perlin" joint="vl5" start="0" end="100">
  <bmlt:parameter name="basefreqx" value="0.5"/>
  <bmlt:parameter name="baseamplitudex" value="0.05"/>
</bmlt:noise>
```

Perlin noise behaviors have the following parameters:

- `basefreqx`, `basefreqy`, `basefreqz`
- `baseamplitudex`, `baseamplitudey`, `baseamplitudez`
- `persistencex`, `persistencey`, `persistencez`

Their values can be set using `bmlt:parameter`.

C.1.8 `<bmlt:facemorph>`

Directly controls a face morph target defined on the avatar mesh.

Attribute	Type	Use	Description
targetname	string	required	name of morph target
intensity	float	required	amount to which the morph target is activated

Table C.8: Attributes of the `<bmlt:facemorph>` element.

BML Example 42 Example of `bmlt:noise`.

```
<facemorph id="face1" targetname="Body_NG-mesh-morpher-Smile01-9" start="0"
intensity ="0.5" end="5"/>
```

C.1.9 `<bmlt:interrupt>`

The `bmlt:interrupt` behavior provides the capability of specifying precisely when specific running or scheduled behaviors should be interrupted.

Attribute	Type	Use	Description
target	string	required	id of the target BML block to interrupt
include	comma separated list of strings	optional	set of behaviors to interrupt in the target BML block (unspecified is all)
exclude	comma separated list of strings	optional	set of behaviors NOT to interrupt in the target BML block (unspecified is none)

Table C.9: Attributes of the `<bmlt:interrupt>` element.

BML Example 43 Interrupt all behaviors in BML, except bml1:speech1 and bml1:gesture1.

```
<bmlt:interrupt id="i1" target="bml1" start="shake1:stroke"
  exclude="speech1,gesture1"/>
```

C.1.10 <bmlt:parametervaluechange>

The parametervaluechange behavior allows the modification of certain parameter values of ongoing behavior.

Attribute	Type	Use	Description
target	string	required	id of the behavior the parametervaluechange targets
paramId	string	required	id of the targeted parameter

Table C.10: Attributes of the `<bmlt:parametervaluechange>` element.

BML Example 44 Change the volume of bml1:speech1.

```
<bmlt:parametervaluechange target="bml1:speech1" paramId="volume"
  start="bml1:speech1:sync1" stroke="bml1:speech1:sync1+1">
  <bmlt:trajectory type="linear" targetValue="90"/>
</bmlt:parametervaluechange>
```

The `bmlt:parametervaluechange` behavior contains a `bmlt:trajectory` element that specifies the trajectory and value of the parameter that is changed.

The `bmlt:parametervaluechange` behavior sets the parameter value on the target behavior according to the trajectory, in such a way that the targetValue is reached at the stroke of the `bmlt:parametervaluechange` behavior. It is not allowed to constrain the end sync of the `bmlt:parametervaluechange` behavior; it ends automatically when the target value is achieved.

Attribute	Type	Use	Description
targetValue	float	required	target value of the parameter
type	string	required	type of the trajectory (e.g. linear, instant)
initialValue	float	optional	start value of the parameter, if omitted the start value is obtained from the targeted behavior

Table C.11: Attributes of the `<bmlt:trajectory>` element.

C.2 Pre-planning and Activation

Scheduling a BML block typically takes a non-negligible amount of time, especially if the timing of speech is to be obtained through speech synthesis software. This is problematic for developing highly responsive virtual humans. BML^T provides *pre-planning* as a mechanism to construct a behavior plan that can be activated later on. In a typical usage scenario of pre-planning, the SAIBA Behavior Planner already knows what behavior to execute, and wants to have it ready for (near) instant execution, for example in reaction to some event such as an incoming response from the user. Pre-planning is set up for a BML block, using the BML^T pre-plan attribute in that block. Pre-planned BML blocks can be activated using a BML^T activate behavior. The pre-planned BML block is activated as soon the activate behavior starts.

BML Example 45 Pre-plan bml1.

```
<bml id="bml1" bmlt:preplan="true">
...
</bml>
```

C.2.1 `<bmlt:activate>`

The `bmlt:activate` behavior activates a pre-planned block.

Attribute	Type	Use	Description
target	string	required	id of the target block to activate

Table C.12: Attributes of the `<bmlt:activate>` element.

BML Example 46 Activate pre-planned block bml1.

```
<bmlt:activate id="a1" target="bml1"/>
```

C.3 Synchronization to Predicted Events

BML^{T} allow synchronization to external, possibly predicted events. This does not require an extension of BML, other than allowing a synchronization to reference ids that are not in the BML script.

BML Example 47 Link conducting beat 1, 2 and 3 of a procedural conducting to tick 1, 2 and 3 of a metronome.

```
<bmlt:procanimation id="conduct1" name="3-beat"/>
<constraint id="c1">
  <synchronize ref="conduct1:start">
    <sync ref="anticipators:metronome1:tick1"/>
  </synchronize>
  <synchronize ref="conduct1:beat2">
    <sync ref="anticipators:metronome1:tick2"/>
  </synchronize>
  <synchronize ref="conduct1:beat3">
    <sync ref="anticipators:metronome1:tick3"/>
  </synchronize>
</constraint>
```

C.4 Persistent BML^{T} behaviors

Some BMLT behaviors are persistent (currently just `bmlt:controller`). They adhere to the same persistent patterns as gaze and posture in core BML.

BML Example 48 Specification of a persistent balance controller.

```
<bml id="bml1" xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
   <bmlt:controller id="balance1" class="BalanceController" start="1"/>
</bml>
```

C.5 Mutually exclusive behavior using replacement groups

Each BML behavior can be assigned a replacement group using a `bmlt:parameter`. Elckerlyc ensures that at most one behavior of each replacement group is played at

the same time. The played behavior is the behavior with the most recent start time. BML^T behaviors can overwrite persistent core BML behaviors by using their name

BML Example 49 Balances with stretched knees (pelvisheight = 1.2m) from time = 0 till time = 4, balances with bend knees (pelvisheight = 0.8m) from time = 4 till time = 8, then balances with stretched knees again.

```
<bml id="bml1" xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
  <bmlt:controller id="balance1" class="BalanceController">
    <bmlt:parameter name="pelvisheight" value="1.2"/>
    <bmlt:parameter name="replacementgroup" value="balance"/>
  </bmlt:controller>
  <bmlt:controller id="balance1" class="BalanceController" start="4" end="8">
    <bmlt:parameter name="pelvisheight" value="0.8"/>
    <bmlt:parameter name="replacementgroup" value="balance"/>
  </bmlt:controller>
</bml>
```

as replacement group.

BML Example 50 A `bmlt:controller` overwriting a persistent `posture` behavior.

```
<bml id="bml1" xmlns:bmlt="http://hmi.ewi.utwente.nl/bmlt">
  <posture id="pose1" stance="standing" shape="open" part="lower"/>
  <bmlt:controller id="balance1" class="BalanceController" start="4">
    <bmlt:parameter name="pelvisheight" value="0.6"/>
    <bmlt:parameter name="replacementgroup" value="posture"/>
  </bmlt:controller>
</bml>
```

C.6 BML^T description extensions

Most BML^T behaviors may also be used as a description extension for a core BML behavior. For example, `bmlt:procanimation` can be used as description extension for `gesture` and `bmlt:controller` can be used as description extension for `posture`.

BML Example 51 A `bmlt:procanimation` behavior used as a description extension for a `gesture`.

```
<gesture start="3" id="beat1" type="LEXICALIZED" lexeme="conduct-3-beat"
  hand="RIGHT">
  <description priority="1" type="bmlt:procanimation">
    <bmlt:procanimation id="beat1" name="conduct-3-beat" start="3">
      <bmlt:parameter name="a" value="5"/>
      <bmlt:parameter name="hand" value="RIGHT"/>
    </bmlt:procanimation>
  </description>
</gesture>
```

C.7 Speech Description Extensions Implemented by Elckerlyc

BML^{T} supports several speech description extensions, including SSML, Microsoft SAPI and various MaryTTS specifications.

BML Example 52 A description extension of `speech`, using Microsoft Speech API.

```
<bml id="bml1">
  <speech id="speech1" start="5">
    <text>I'm speaking BML.</text>
    <description priority="1" type="application/msapi+xml">
      <sapi>I'm speaking <spell>BML</spell>.</sapi>
    </description>
  </speech>
</bml>
```

BML Example 53 A description extension of `speech`, using SSML.

```
<bml id="bml1">
  <speech id="speech1" start="5">
    <text>I'm speaking BML.</text>
    <description priority="1" type="application/ssml+xml">
      <speak xmlns="http://www.w3.org/2001/10/synthesis">
      I'm <prosody pitch="high">speaking</prosody> BML.
      </speak>
    </description>
  </speech>
</bml>
```

BML Example 54 A description extension of `speech`, Mary TTS.

```
<bml id="bml1">
  <speech id="speech1" start="5">
    <text>I'm speaking BML.</text>
    <description priority="10" type="maryxml">
      <maryxml xmlns="http://mary.dfki.de/2002/MaryXML">
      I'm speaking BML.
      </maryxml>
    </description>
  </speech>
</bml>
```

C.8 BML^T Feedback

In addition to the feedback specified in core BML, Elckerlyc provides global time stamps in all its feedback messages and additionally a local time stamp (time in seconds since start of the BML performance) for Sync-Point Progress Feedback. Elckerlyc also provides information on the scheduling status of a BML block, using Planning Start and Planning Finished feedback.

C.8.1 Scheduling Start Feedback

Notifies the behavior planner that scheduling of a requested BML performance has begun. The message includes:

- a BML block identifier
- a global timestamp
- the predicted start time of the BML block

C.8.2 Scheduling Finished Feedback

Notifies the behavior planner that scheduling of a requested BML block is finished. The message includes:

- a BML block identifier
- a global timestamp
- the predicted start time of the BML block

C.9 The BML^T BML attributes

In addition to the `preplan` attribute, BML^T provides several other attributes that are applied in the `bml` element itself.

C.9.1 append-after

append-after(X) scheduling attribute instructs the Realizer to execute the new BML block immediately after all behaviors from the prior BML blocks on list X have finished.

BML Example 55 `bml4` is to be appended after `bml2` and `bml3`.

```
<bml id="bml4" scheduling="append-after(bml2,bml3)">
...
</bml>
```

C.9.2 allowexternalrefs

The `allowexternalrefs` attribute is used to indicate that a BML block may contain time constraints that refer to behaviors in other (external) BML blocks. Such references are of the form bmlid:behaviorid:syncid.

BML Example 56 A `bmlt:parametervaluechange` that refers to `speech1` in bml block `bml1`.

```
<bml id="bml2" bmlt:allowexternalrefs="true">
  <bmlt:parametervaluechange id="pvc1" target="bml1:speech1"
    paramId="volume" start="bml1:speech1:sync1" stroke="bml1:speech1:end">
    <bmlt:trajectory type="linear" initialValue="0" targetValue="100"/>
  </bmlt:parametervaluechange>
</bml>
```

C.9.3 The interrupt shorthand

The interrupt attribute is a shorthand for the SAIBA Behavior Planner to remove a selected set of BML blocks from the multimodal behavior plan before scheduling the content of the BML block it is in.

BML Example 57 Interrupt all behaviors in `bml1`, `bml2`, .., `bmln` before scheduling `bmlNew`. Top: the BML^{T} shorthand; bottom: an outline of the SAIBA Behavior Planner functionality it represents.

(a) Interrupt shorthand.

```
<bml id="bmlNew" bmlt:interrupt="bml1,bml2,..,bmln">
  bmlNew content
</bml>
```

(b) SAIBA Behavior Planner functionality implemented by the interrupt shorthand.

1. Send a BML block to the Realizer that interrupts `bml1`..`bmln`:

```
<bml id="bmlInterrupt">
    <bmlt:interrupt id="interrupt1" target="bml1"/>
    <bmlt:interrupt id="interrupt2" target="bml2"/>
    ..
    <bmlt:interrupt id="interruptn" target="bmln"/>
</bml>
```

2. Wait for block end feedback of bmlInterrupt (to make sure `bml1`..`bmln` are properly removed from the multimodal behavior plan).
3. Send the new BML block bmlNew:

```
<bml id="bmlNew">
    bmlNew content
</bml>
```

SIKS Dissertation Series

Since 1998, all dissertations written by Ph.D. students who have conducted their research under auspices of a senior research fellow of the SIKS research school are published in the SIKS Dissertation Series. This thesis is the 322th in the series.

2011-23 Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media*
2011-22 Junte Zhang (UVA), *System Evaluation of Archival Description and Access*
2011-21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems*
2011-20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach*
2011-19 Ellen Rusman (OU), *The Mind's Eye on Personal Profiles*
2011-18 Mark Ponsen (UM), *Strategic Decision-Making in complex games*
2011-17 Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness*
2011-16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity*
2011-15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*
2011-14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets*
2011-13 Xiaoyu Mao (UvT), *Airport under Control. Multi-agent Scheduling for Airport Ground Handling*
2011-12 Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining*
2011-11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective*
2011-10 Bart Bogaert (UvT), *Cloud Content Contention*
2011-09 Tim de Jong (OU), *Contextualised Mobile Media for Learning*
2011-08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues*
2011-07 Yujia Cao (UT), *Multimodal Information Presentation for High-Load Human Computer Interaction*
2011-06 Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage*
2011-05 Base van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline*
2011-04 Hado van Hasselt (UU), *Insights in Reinforcement Learning Formal analysis and empirical evaluation of temporal-difference learning algorithms*
2011-03 Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems*
2011-02 Nick Tinnemeier (UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language*
2011-01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*

2010-53 Edgar Meij (UVA), *Combining Concepts and Language Models for Information Access*
2010-52 Peter-Paul van Maanen (VU), *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*
2010-51 Alia Khairia Amin (CWI), *Understanding and supporting information seeking tasks in multiple sources*
2010-50 Bouke Huurnink (UVA), *Search in Audiovisual Broadcast Archives*
2010-49 Jahn-Takeshi Saito (UM), *Solving difficult game positions*
2010-48 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets*
2010-47 Chen Li (UT), *Mining Process Model Variants: Challenges, Techniques, Examples*
2010-46 Vincent Pijpers (VU), *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*
2010-45 Vasilios Andrikopoulos (UvT), *A theory and model for the evolution of software services*
2010-44 Pieter Bellekens (TUE), *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*
2010-43 Peter van Kranenburg (UU), *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*
2010-42 Sybren de Kinderen (VU), *Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach*
2010-41 Guillaume Chaslot (UM), *Monte-Carlo Tree Search*
2010-40 Mark van Assem (VU), *Converting and Integrating Vocabularies for the Semantic Web*
2010-39 Ghazanfar Farooq Siddiqui (VU), *Integrative modeling of emotions in virtual agents*
2010-38 Dirk Fahland (TUE), *From Scenarios to components*
2010-37 Niels Lohmann (TUE), *Correctness of services and their composition*
2010-36 Jose Janssen (OU), *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification*
2010-35 Dolf Trieschnigg (UT), *Proof of Concept: Concept-based Biomedical Information Retrieval*
2010-34 Teduh Dirgahayu (UT), *Interaction Design in Service Compositions*
2010-33 Robin Aly (UT), *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*
2010-32 Marcel Hiel (UvT), *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*
2010-31 Victor de Boer (UVA), *Ontology Enrichment from Heterogeneous Sources on the Web*

2010-30 Marieke van Erp (UvT), *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval*
2010-29 Stratos Idreos(CWI), *Database Cracking: Towards Auto-tuning Database Kernels*
2010-28 Arne Koopman (UU), *Characteristic Relational Patterns*
2010-27 Marten Voulon (UL), *Automatisch contracteren*
2010-26 Ying Zhang (CWI), *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*
2010-25 Zulfiqar Ali Memon (VU), *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*
2010-24 Dmytro Tykhonov, *Designing Generic and Efficient Negotiation Strategies*
2010-23 Bas Steunebrink (UU), *The Logical Structure of Emotions*
2010-22 Michiel Hildebrand (CWI), *End-user Support for Access to*
Heterogeneous Linked Data
2010-21 Harold van Heerde (UT), *Privacy-aware data management by means of data degradation*
2010-20 Ivo Swartjes (UT), *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*
2010-19 Henriette Cramer (UvA), *People's Responses to Autonomous and Adaptive Systems*
2010-18 Charlotte Gerritsen (VU), *Caught in the Act: Investigating Crime by Agent-Based Simulation*
2010-17 Spyros Kotoulas (VU), *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*
2010-16 Sicco Verwer (TUD), *Efficient Identification of Timed Automata, theory and practice*
2010-15 Lianne Bodenstaff (UT), *Managing Dependency Relations in Inter-Organizational Models*
2010-14 Sander van Splunter (VU), *Automated Web Service Reconfiguration*
2010-13 Gianluigi Folino (RUN), *High Performance Data Mining using Bio-inspired techniques*
2010-12 Susan van den Braak (UU), *Sensemaking software for crime analysis*
2010-11 Adriaan Ter Mors (TUD), *The world according to MARP: Multi-Agent Route Planning*
2010-10 Rebecca Ong (UL), *Mobile Communication and Protection of Children*
2010-09 Hugo Kielman (UL), *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging*
2010-08 Krzysztof Siewicz (UL), *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*
2010-07 Wim Fikkert (UT), *Gesture Interaction at a Distance*
2010-06 Sander Bakkes (UvT), *Rapid Adaptation of Video Game AI*
2010-05 Claudia Hauff (UT), *Predicting the Effectiveness of Queries and Retrieval Systems*
2010-04 Olga Kulyk (UT), *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*
2010-03 Joost Geurts (CWI), *A Document Engineering Model and Processing Framework for Multimedia documents*
2010-02 Ingo Wassink (UT), *Work flows in Life Science*
2010-01 Matthijs van Leeuwen (UU), *Patterns that Matter*
2009-46 Loredana Afanasiev (UvA), *Querying XML: Benchmarks and Recursion*
2009-45 Jilles Vreeken (UU), *Making Pattern Mining Useful*
2009-44 Roberto Santana Tapia (UT), *Assessing Business-IT Alignment in Networked Organizations*
2009-43 Virginia Nunes Leal Franqueira (UT), *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*
2009-42 Toine Bogers, *Recommender Systems for Social Bookmarking*
2009-41 Igor Berezhnyy (UvT), *Digital Analysis of Paintings*
2009-40 Stephan Raaijmakers (UvT), *Multinomial Language Learning: Investigations into the Geometry of Language*
2009-39 Christian Stahl (TUE, Humboldt–Universitaet zu Berlin), *Service Substitution – A Behavioral Approach Based on Petri Nets*
2009-38 Riina Vuorikari (OU), *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context*
2009-37 Hendrik Drachsler (OUN), *Navigation Support for Learners in Informal Learning Networks*
2009-36 Marco Kalz (OUN), *Placement Support for Learners in Learning Networks*
2009-35 Wouter Koelewijn (UL), *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*
2009-34 Inge van de Weerd (UU), *Advancing in Software Product Management: An Incremental Method Engineering Approach*
2009-33 Khiet Truong (UT), *How Does Real Affect Affect Affect Recognition In Speech?*
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU), *Architectural Knowledge Management: Supporting Architects and Auditors*
2009-31 Sofiya Katrenko (UVA), *A Closer Look at Learning Relations from Text*
2009-30 Marcin Zukowski (CWI), *Balancing vectorized query execution with bandwidth-optimized storage*
2009-29 Stanislav Pokraev (UT), *Model-Driven Semantic Integration of Service-Oriented Applications*
2009-28 Sander Evers (UT), *Sensor Data Management with Probabilistic Models*
2009-27 Christian Glahn (OU), *Contextual Support of social Engagement and Reflection on the Web*
2009-26 Fernando Koch (UU), *An Agent-Based Model for the Development of Intelligent Mobile Services*
2009-25 Alex van Ballegooij (CWI), *RAM: Array Database Management through Relational Mapping*
2009-24 Annerieke Heuvelink (VUA), *Cognitive Models for Training Simulations*
2009-23 Peter Hofgesang (VU), *Modelling Web Usage in a Changing Environment*
2009-22 Pavel Serdyukov (UT), *Search For Expertise: Going beyond direct evidence*
2009-21 Stijn Vanderlooy (UM), *Ranking and Reliable Classification*
2009-20 Bob van der Vecht (UU), *Adjustable Autonomy: Controling Influences on Decision Making*
2009-19 Valentin Robu (CWI), *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*
2009-18 Fabian Groffen (CWI), *Armada, An Evolving Database System*
2009-17 Laurens van der Maaten (UvT), *Feature Extraction from Visual Data*
2009-16 Fritz Reul (UvT), *New Architectures in Computer Chess*
2009-15 Rinke Hoekstra (UVA), *Ontology Representation - Design Patterns and Ontologies that Make Sense*
2009-14 Maksym Korotkiy (VU), *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*
2009-13 Steven de Jong (UM), *Fairness in Multi-Agent Systems*
2009-12 Peter Massuthe (TUE, Humboldt–Universitaet zu Berlin), *perating Guidelines for Services*

2009-11 Alexander Boer (UVA), *Legal Theory, Sources of Law and the Semantic Web*
2009-10 Jan Wielemaker (UVA), *Logic programming for knowledge-intensive interactive applications*
2009-09 Benjamin Kanagwa (RUN), *Design, Discovery and Construction of Service-oriented Systems*
2009-08 Volker Nannen (VU), *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*
2009-07 Ronald Poppe (UT), *Discriminative Vision-Based Recovery and Recognition of Human Motion*
2009-06 Muhammad Subianto (UU), *Understanding Classification*
2009-05 Sietse Overbeek (RUN), *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality*
2009-04 Josephine Nabukenya (RUN), *Improving the Quality of Organisational Policy Making using Collaboration Engineering*
2009-03 Hans Stol (UvT), *A Framework for Evidence-based Policy Making Using IT*
2009-02 Willem Robert van Hage (VU), *Evaluating Ontology-Alignment Techniques*
2009-01 Rasa Jurgelenaite (RUN), *Symmetric Causal Independence Models*