

**ViSiCAST Deliverable D4-2:
SiGML Notation-Avatar Software Driver**

Project Number:	IST-1999-10500
Project Title:	ViSiCAST Virtual Signing: Capture, Animation, Storage and Transmission
Deliverable Type:	Int
Deliverable Number:	D4-2
Contractual Date of Delivery:	December 2001
Actual Date of Delivery:	December 2001
Title of Deliverable:	SiGML Notation-Avatar Software Driver
Work-Package contributing to the Deliverable:	Workpackage 4 (Animation and Modelling)
Nature of the Deliverable:	PR (Prototype)
Author(s):	Ralph Elliott, John Glauert, Richard Kennaway, Kevin Parsons (UEA)

Abstract:

This document describes prototype synthetic deaf signing animation software. This software effectively forms the second half of the English-text-to-synthetic-signing pipeline described in the opening chapter of ViSiCAST deliverable D5-1. It takes as input a description of a sequence of signing gestures expressed in SiGML, from which it generates a corresponding stream of avatar animation parameters, which it then uses to drive a signing avatar on-screen.

1 Introduction

This deliverable consists of prototype software which generates synthetic deaf signing via a signing avatar, given a description of the required sequence of signing gestures expressed in the SiGML notation. The present covering document briefly describes this software and its role in the ViSiCAST project.

A major objective of ViSiCAST is the development of a prototype text-to-signing system, which takes (English) natural language text and translates this text into authentic sign language, represented in SiGML, which is then used to drive a computer-generated avatar, or virtual human, thus enabling the content of the original text to be presented to deaf users in their preferred language. The function and architecture of this text-to-signing system are described in some detail in Chapter 1 of deliverable D5-1, *Interface Definitions*, where the system architecture is presented as a pipeline of processing stages. Responsibility for the implementation of this pipeline is shared between the project's two core technology workpackages, namely WP5, *Language and Notation*, and WP4, *Animation and Modelling*. WP5 is responsible for the first half of the processing pipeline, and WP4 for the second half. The interface between the two halves is SiGML, the *Signing Gesture Markup Language* developed within the project, whose first version was defined in deliverable D5-2. SiGML is an XML application based on *HamNoSys*, the Hamburg Notation System, a well-established notation for representing sign languages at (approximately) the phonological level. More specifically, SiGML is based on HamNoSys version 4, which was developed in an earlier phase of the project, and which is described in deliverable D5-1. Although the text-to-signing pipeline provides the immediate context for the present deliverable, it should be emphasised that the SiGML-to-signing module for which the deliverable provides the initial prototype is intended to provide a general vehicle for applications requiring on-screen signing.

The next section (Section 2) describes the software of this deliverable in more detail. The last section (Section 3) describes the packaging and deployment of the software.

2 Overview of D4-2 Software

The software for this deliverable consists of two modules, the core animation engine and a collection of integration ("glue") components, which together result in a single package which displays the appropriate avatar and (repeatedly) allows the user to specify a SiGML input, from which a stream of animation parameters is then generated and fed to the avatar for presentation to the user. (The generated animation parameter stream is also saved in a file for future use.) In this section we describe these two modules in more detail.

2.1 *Animgen*

A prototype implementation of the first half of the text-to-signing pipeline was supplied as part of deliverable D5-3. Thus the function of the second half of the pipeline, as represented in the present deliverable, is to provide software to animate the set of example SiGML sequences generated by the D5-3 system. The core of the deliverable is a software module called *animgen*, a synthetic animation engine which converts a SiGML input stream into a stream of avatar animation parameters. These animation parameters may take either of two forms, corresponding to the two distinct target avatar technologies of interest. These are: the proprietary avatar technology supplied by Televirtual Ltd. (one of the ViSiCAST partners), and the VRML-based H-Anim avatar.

The first of these is ViSiCAST's primary signing avatar technology: it is more advanced than H-Anim, giving both high visual quality and high performance when supported by appropriate GPU (graphics processing unit) hardware. In this technology the avatar's appearance is defined by a complex seamless deformable mesh constructed from a large number of small textured polygons. The physical configuration of the mesh is dynamically controlled by the configuration of an underlying "skeleton", a set of bones (and muscles) organised, on anatomical lines, as a tree-structured hierarchy. Hence, a pose of the avatar is determined by specifying the spatial position and orientation of each bone/muscle (and, in the case of a muscle, also its length). So, a time-stamped sequence of animation parameter sets of this kind is sufficient, in principle, to drive the avatar.

This technology is realised as a collection of data files, defining the skeleton, mesh, and other details of a particular virtual human, together with the software modules needed to "drive" an animation of the model. The technology was first developed in the context of a system driven by data originating from motion-capture devices. Towards the end of the first year of the project (2000), Televirtual made this avatar technology available to partners in the form of an ActiveX Control, suitable for deployment in various software contexts, in particular in a WWW-browser, and including the data files defining the visual features of a project-specific avatar called *Visia*. Although still intended primarily to support applications driven by motion capture data, this software provides a rudimentary interface allowing the configuration of the *Visia* avatar to be set, as just described, in accordance with directly supplied skeleton configuration data, a *Bones Animation Format (BAF)* frame. The BAF animation data stream format was first used within ViSiCAST as the transmission format (with compression) for the first deliverable of the project's TV/Broadcast workpackage (WP1, deliverable D1-1, *Direct-Sign*). The *Visia* avatar, thus packaged as an ActiveX Control formed a major component of deliverable D2-1, *ViSiCAST Avatar Controls*.

For development purposes, however, it often proves more convenient to use *animgen* in conjunction with the alternative, H-Anim-based, target. In particular, as part of the *animgen* development effort a simple "stick-man" avatar has been defined. This stick-man is H-Anim compliant, although it also incorporates movable eyebrows (albeit in a crude form), in order to support the very restricted set of non-manual SiGML features used in the D5-3 examples. This avatar is thus similar to Televirtual's *Visia* in as much as each static pose is determined by the configuration of a hierarchically structured skeleton. Hence, animation of this avatar is defined in a manner very similar to that described above for *Visia*, that is, by a time-stamped sequence of skeleton configuration parameter sets — although for the H-Anim avatar these data streams take the form of VRML data-structures suitable for use by the VRML animation functions.

The main advantages of using the stick-man avatar as target in a development context are that animation data streams for this avatar are somewhat more compact, and quicker to generate, than for *Visia*, and the resulting animations, while obviously less visually appealing, are sometimes easier to inspect and to evaluate from a technical (as opposed to an aesthetic) perspective. A further significant factor is that the VRML animation model has well-defined timing properties, whereas the rudimentary interface mentioned above, which is the only currently available means of driving *Visia* from a BAF-stream, merely provides a method allowing a "frame" (i.e. a static pose in the sequence) to be passed to the avatar for rendering, but does not provide any means of controlling (or indeed of monitoring) the precise time at which the requested rendering is actually accomplished. Hence, while it is perfectly capable of giving an approximate indication of behaviour, the currently available version of *Visia* is not a suitable vehicle for proper evaluation of synthetically generated animation. However, the enhancement of the *Visia* ActiveX Control to include a BAF-frame rendering interface with well-defined timing properties is a priority for the project, and is expected to be available very shortly, allowing its incorporation in milestone M5-11 which marks the integration of the two halves of the text-to-signing pipeline (scheduled for achievement by the end of February 2002).

As stated at the start of this section, the development of *animgen* for the present deliverable has focussed on support for those SiGML features used in the set of examples generated by the D5-3 natural language processing software. This prototype version of *animgen* thus does not yet implement all features of SiGML, and in particular, its support for non-manual features of SiGML is restricted to the very small subset (eyebrow movements) required for the D5-3 examples. Also, a small range of manual features are not yet supported. For a full list of features not currently supported the reader is referred to Appendix B. Although the version of SiGML supported is essentially that delivered in D5-2, a small number of minor modifications have been made in the light of subsequent experience with features introduced with version 4 of HamNoSys. The current version of the SiGML DTD, incorporating these modifications, may be found in Appendix A.

The objective of synthesising animations for all the D5-3 sentences has almost but not quite been met. The set consists of a sequence of 21 sentences, containing a total of 81 signs. Of these 81 signs, *animgen* is unable to synthesise anything more useful than a “NULL” marker for the following:

```
glass green plate.
```

In addition, a further 5 signs out of the 81 contain some feature which *animgen* does not support, leading to partially correct synthesis for each of these:

```
big soup_bowl teaspoon large small.
```

Animgen is currently implemented in Perl. In consequence it is relatively inefficient: on a high-performance PC (with a processor speed in excess of 1.5GHz), it generates BAF frames at a rate of at least 15 fps, the rate generally taken within the project as being the minimum acceptable for adequate quality of signing animation). For the H-Anim stick-man target, the frame rate improves by a factor of at least 2. However, we intend to recode the current Perl implementation, which contains several obvious Perl-dependent inefficiencies, in a compiled language such as C++ or Java. We expect this to improve performance by a factor of at 3 or 4 at worst, and more probably by an order of magnitude. For technical details of *animgen*'s implementation of the mapping from SiGML documents to streams of animation parameters, the reader is referred to Appendix E, which contain copies of a paper presented at GW2001, the most recent Gesture Workshop (London, April 2001).

2.2 Integration (“Glue”) Components

The integration software consists of a collection of COM components which connect the relevant sub-stages of the pipeline and provide a simple graphical user interface to the signing avatar. These components support the construction of simple applications (e.g. in VB) or HTML pages (with JavaScript) which present animations of signing sequences generated by *animgen*, using either the VRML stick-man or Visia. The standard input format for *animgen* is SiGML, but using the HamNoSys-to-SiGML converter included with deliverable D5-3, these components also allow HNST (HamNoSys token sequences) as an acceptable alternative input format. In practice, as the standard output format for the English-to-BSL software developed at UEA for D5-3 is HNST, this latter is often the most convenient input format for the animation software described here.

The architecture and function of this integration software is described in more detail in a separate report, included here as Appendix D.

3 Packaging and Deployment of D4-2 Software

An installer is supplied for the delivered software (which runs on PCs under Windows XP/2000/98). This installer assumes (and checks for) prior installations of the following supporting software

components: Java JRE or JDK, v1.3 (needed by the HamNoSys-to-SiGML translation software), ActivePerl, v5.6 (needed by *animgen*), Cortona VRML Player/Plugin, v3.1 (needed to play VRML-format animations), ViSiCAST Avatar Controls (needed for the Visia avatar which plays BAF-format animations). The README text file supplied with the installer provides the necessary details about the installation process. The installer provides two ways for the end user to access the software, corresponding to the two supported animation formats described in the previous section:

- An HTML page which allows the user to generate and play VRML-format animations using the stick-man figure described in the previous section.
- A stand-alone application which provides analogous facilities for BAF-format animations using the project's Visia avatar.¹

For the purposes of the present deliverable and its evaluation we regard the first of these two as the primary (“reference”) implementation, due to the absence of BAF-frame synchronisation features in the current Visia avatar used by the second, as described in Section 2.1. Nevertheless, we include the Visia-based application in the deliverable distribution as well, in order to demonstrate the effectiveness of the integration software, and as a preliminary guide to what may be expected in the near future.

The installer inserts *Start* menu items allowing the user to invoke either of these configurations of the software. In either case the user may specify the folder containing data files, and may then chose any one of three input formats:

- HNST (HamNoSys tokens);
- SiGML;
- Avatar Animation Parameters (BAF or VRML).

In each case the *Files* panel displays a list of available files in the given format. If a file is specified using this panel, a click on the *Play* button will cause the animation for the specified file to be played by the avatar. For HNST or SiGML input, before playing, the animation data is first generated using the HamNoSys-to-SiGML converter, and *animgen*.² In both cases, the frame rate for the generated animation data may be specified using the combo box, and the resulting data stream is saved in a file which is fed as input to the avatar player software; this file is also available for subsequent use (as may be seen by switching to BAF/VRML input format).³

The VRML stick-man avatar comes with a simple set of GUI features providing information to the user, and allowing the user a useful measure of interactive control of the animation. The user documentation for these features may be found in Appendix C.

The installed implementation folder contains a handful of example files, giving animation sequences for: the full set of D5-3 examples, a much shorter sequence of three sentences from this set, the first sentence ever handled by the text-to-signing software (“I take the mug.”), and a simple sequence of eyebrow manipulations. Obviously, the user is free to add further SiGML (or HNST) files to this collection. Note that when a sign contains features not currently supported by *animgen*, that sign is replaced either by a “NULL” place-holder, or by a partially correct sign, in the final animation.

¹ An HTML page for BAF-format animations using Visia, analogous the that for VRML-format animations, is also available, but for simplicity's sake is not included in this distribution.

² The unwary user is warned that the generation of the animation parameters, especially in BAF format, takes a noticeable amount of time. (See Section 2.1 for more details.)

³ As indicated in Appendix D, the integration software is capable of streaming BAF animation data directly to the Visia avatar via an internal cache without using an intermediate file; this facility is used in the initial prototype of the fully integrated pipeline (Milestone 5-11, but not included in the present deliverable).

Further information about *animgen*'s treatment of unsupported SiGML features, and a list of signs in the D5-3 example set containing such features was given in Section 2.2 above. *Animgen* maintains a logfile, `PerlWrapperLog.txt` (in the implementation folder), which the interested user may consult for details of its progress.

References to Other ViSiCAST Deliverables

D2-1 *Internet Browser Plugin* (March 2001)

D5-1 *Interface Definitions* (March 2001)

D5-2 *SiGML Definition* (May 2001)

D5-3 *Prototype Text to Sign Notation* (August 2001)

Appendices

A SiGML DTD at 2001-12

B Status of AnimGen's Implementation of SiGML Features

C *Animgen* VRML GUI Widgets

D Report: *HamNoSys to Animation Components* (Kevin Parsons)

E Report: *Synthetic Animation of Deaf Signing Gestures* (Richard Kennaway)

Appendix A (D4-2) SiGML DTD at 2001-12

Top-Level SiGML DTD

```
<!-- sigmlh4.dtd

2001-07-27:
Introduction of sigmlh4 DTD, with changes arising from the
conversion of the h2s translator to process HamNoSys 4.
(RE)
-use sigmlh4manual rather than sigmlmanual;

This is the DTD for the top level of core SiGML 1.0, describing
how to combine various sources of signing information into a
single stream.
The HamNoSys-based representations of manual and non-manual
gestures, so-called core SiGML 1.0, are split off into separate
files, sigmlmanual.dtd and sigmlnonmanual.dtd,
which are included in this one.
-->

<!--
We use a SMIL-like representation of for a sign's speed attribute,
which has a numeric value with 1.0 as the implicit standard value.
-->
<!ENTITY % speed
    "speed CDATA #IMPLIED"
>

<!--
A gloss for a sign is an arbitrary string, naming that sign.
The language of the gloss word, and the sign language of the
glossed sign may also be specified (the former by its standard
ISO code).
-->

<!ENTITY % gloss_attribs
    "gloss_language CDATA #IMPLIED
    sign_language CDATA #IMPLIED"
>

<!ENTITY % glossoption_attribs
    "gloss CDATA #IMPLIED
    %gloss_attribs;"
>

<!--
Each SiGML sign is one of:

(0) Motion capture data (a Televirtual ".all" data file)
(1) Bone-set animation data (Televirtual "BAF" data format)
(2) A core SiGML gestural description (based on HamNoSys 4)
(3) A gloss, identifying a sign defined elsewhere

Whatever its kind, a sign may have the standard gloss attributes.
```

A SiGML segment is a single sign or a gestural segment, that is, a sequence of gestural signs accompanied by a single unit of non-manual signing.

-->

```
<!ENTITY % sigml_sign
  "mocap_sign |
  bonesanimation_sign |
  hamgestural_sign |
  gloss_sign"
>
```

```
<!ENTITY % sigml_segment
  "%sigml_sign; |
  hamgestural_segment"
>
```

<!--

A SiGML document consists of an optional signing avatar characterization followed by a sequence of signing segments.

-->

```
<!ELEMENT sigml ( avatar?, ( %sigml_segment; )* )>
```

<!--

A signing avatar may have a name, as used by Televirtual avatar software (e.g. "visia"), and a URI for any relevant avatar description data (e.g. for a VRML avatar).

-->

```
<!ELEMENT avatar EMPTY>
<!ATTLIST avatar
  name CDATA #IMPLIED
  definition_uri CDATA #IMPLIED
>
```

<!--

(0) A Motion capture data sign is specified by the name of the motion capture data (.all) file (excluding the suffix). Eventually this should be a URI, but Televirtual currently require these files to be in an avatar-specific directory.

-->

```
<!ELEMENT mocap_sign EMPTY>
<!ATTLIST mocap_sign
  mocap_uri CDATA #REQUIRED
  %glossoption_attribs;
  %speed;
>
```

<!--

(1) A Bone set animation sign is specified by the location of the time-stamped boneset sequence data (.baf file).

-->

```
<!ELEMENT bonesanimation_sign EMPTY>
<!ATTLIST bonesanimation_sign
```

```

    bonesdata_uri CDATA #REQUIRED
    %glossoption_attribs;
    %speed;
>

<!--
(2) A HamNoSys-style gestural sign is specified by a an optional
manual part and an optional non-manual part.
-->

<!ELEMENT hamgestural_sign
(
    ( sign_manual, sign_nonmanual? )
    |
    ( sign_nonmanual, sign_manual? )
)?
>
<!ATTLIST hamgestural_sign
    %glossoption_attribs;
    %speed;
>

<!--
(3) A gloss sign is defined by the standard gloss attributes, but
in this case the gloss name is required rather than optional.
-->

<!ELEMENT gloss_sign EMPTY>
<!ATTLIST gloss_sign
    gloss CDATA #REQUIRED
    %gloss_attribs;
    %speed;
>

<!--
A gestural segment allows a single unit of non-manual signing
to accompany a (non-trivial) sequence of signing gestures.
If any feature is specified both at the segment level and at
the level of an individual sign within the segment, then the
latter specification takes precedence over the former (or, in
the case of speed, is interpreted relative to the former, as
in SMIL). This applies to the following features:
    -specification for a given non-manual tier;
    -gloss/sign language attributes;
    -speed.
Speed example: if the segment has "speed='2'" and an enclosed
sign has "speed='0.75'", then the effective speed value for
that sign is 1.5.
-->

<!ELEMENT hamgestural_segment
(
    ( hamgestural_sign, hamgestural_sign+, sign_nonmanual )
    |
    ( sign_nonmanual, hamgestural_sign, hamgestural_sign+ )
)
>
<!ATTLIST hamgestural_segment

```

```
    %gloss_attribs;
    %speed;
>

<!--
Include the DTDs for manual and non-manual core SIGML.
-->

<!ENTITY % sigmlmanualdtd SYSTEM "sigmlh4manual.dtd">
%sigmlmanualdtd;

<!ENTITY % sigmlnonmanualdtd SYSTEM "sigmlnonmanual.dtd">
%sigmlnonmanualdtd;

<!-- End of SIGML DTD -->
```

DTD for Manual SiGML

<!-- sigmlh4manual.dtd

2001-08-20 to 2001-08-29:

various changes for HamNoSys 4 (RE):

- add options "dorsal | palmar" to %side entity (NB "radial | ulnar" are already there), and include side attributes in location_hand;
- remove "wristpulse" option from %handpart; (use "wristback" with side="palmar" instead);
- remove several apparently redundant options from %fingerpart; and %side;
- remove JRK's {def,use}_locname attributes from %handconfig_attribs;
- introduce use_locname element, and allow it as an option in %location;
- introduce def_locname attribute into: location_{hand,bodyarm} (where it is currently unused), {directed,circular}motion, wristrotation, changeposture, and %handconfig_attribs;
- replace old wristrotation element with a new wristmotion one (an option for simplemotion), and introduce the associated %wristmotion attribute set;
- enhanced set of %repetition_attribs, as indicated by section 3.5.6 of D5-1;
- remove JRK's brushing attribute from motions, and instead allow directedmotion and circularmotion (but no others for the present) to have an optional %location child, representing the location at which the brushing contact is made (see section 3.5.3 of D5-1);
- introduce alternating, second_alternating attributes into the collection of %motion_attribs; (to be reviewed: logically, these should be _repetition_attributes?, see section 3.5.2 of D5-1, which defines the semantics;)

2001-07-27:

Introduction of sigmlh4manual DTD, with changes arising from the conversion of the h2s translator to process HamNoSys 4. (RE)

- locbody entity changes: rename undernose, mouth, aboves shoulders, as nostrils, lips, shouldertop, and introduce earlobe;
- introduce opposed_finger attribute into %handconfig_attribs, to represent the "large" modifier that may be attached to a finger to indicate that it's the one opposed to the thumb in some hand postures as described at the end of sect. 3.2.2 of D5-1;

This is the DTD for the manual subset of core SigML 1.0,
defining the sign_manual element.

-->

<!-- ##### ENTITY definitions ##### -->

<!ENTITY % boolfalse '(true | false) "false"'

<!-- Directions. -->

<!--

8 compass points, imagined on a vertical circle,
seen from the signer's point of view.

The abbreviations are u=up, d=down, r=right, l=left.

-->

<!ENTITY % the8directions

```
"u
| ur
| r
| dr
| d
| dl
| l
| ul"
```

>

<!--

26 directions, being the faces, edges, and vertices of a cube
centred on the signer, labelled from the signer's point of view.

The abbreviations are u=up, d=down, r=right, l=left, o=out, i=in.

-->

<!ENTITY % the26directions

```
"%the8directions;
| ol
| o
| or
| il
| i
| ir
| uo
| do
| di
| ui
| uol
| dol
| dil
| uil
| uor
| dor
| dir
| uir"
```

>

<!--

4 directions for the major axis of an ellipse, lying in a

vertical circle, seen from the signer's point of view.
The abbreviations are h=horizontal, v=vertical, ur=up/right,
ul=up/left.

-->

```
<!ENTITY % ellipse_direction
  "( h
   | ur
   | v
   | ul ) #IMPLIED"
>
```

<!-- Handshapes. -->

```
<!ENTITY % basichandshape
  "( fist
   | flat
   | finger2
   | finger23
   | finger23spread
   | finger2345
   | pinch12
   | pinchall
   | pinch12open
   | cee12
   | ceeall
   | cee12open) #IMPLIED"
>
```

<!--

```
<!ENTITY % fingerbending
  "( bent
   | round
   | hooked
   | dblbent
   | dblhooked
   | halfbent ) #IMPLIED"
>
```

```
<!ENTITY % thumbpos
  "( out
   | across
   | opposed ) #IMPLIED"
>
-->
```

```
<!ENTITY % ceeopening
  "( tight
   | slack ) #IMPLIED"
>
```

<!-- Locations. -->

```
<!ENTITY % locbody
  " head
   | headtop
   | forehead
   | eyebrows
   | eyes
```

```

| uppereyelid
| lowereyelid
| nose
| nostrils
| lips
| upperlip
| lowerlip
| tongue
| teeth
| upperteeth
| lowerteeth
| chin
| underchin
| neck
| shoulders
| shouldertop
| chest
| stomach
| belowstomach
| ear
| earlobe
| cheek"
>

<!ENTITY % locarm
" upperarm
| elbow
| elbowinside
| lowerarm"
>

<!ENTITY % handpart
" wristback
| thumbball
| palm
| handback
| thumbside
| pinkyside"
>

<!ENTITY % fingerpart
" tip
| nail
| midjoint
| base
| side"
>

<!ENTITY % pcontact
" touch
| close"
>

<!ENTITY % contact_bodyarm
"( %pcontact;
| armextended ) #IMPLIED"
>

<!ENTITY % contact_hand
"( %pcontact;
| interlock

```

```

    | cross ) #IMPLIED"
>

<!ENTITY % side
    "( left_beside
    | left_at
    | right_at
    | right_beside
    | front
    | back
    | dorsal
    | palmar
    | radial
    | ulnar ) #IMPLIED"
>

<!ENTITY % site_bodyarm
    "( %locbody;
    | %locarm;
    | neutralspace ) #IMPLIED"
>

<!ENTITY % site_hand
    "( %handpart;
    | %fingerpart; ) #IMPLIED"
>

<!--
The values of locname are the possible labels that may be
attached to positions for later reference.
-->

<!ENTITY % locname "( 1 | 2 | 3 | 4 | 5 )">

<!ENTITY % def_locname_attrib "def_locname %locname; #IMPLIED">
<!ENTITY % use_locname_attrib "use_locname %locname; #REQUIRED">

<!-- Motion. -->

<!ENTITY % wristmotion
    " nodding
    | swinging
    | twisting
    | stircw
    | stirccw"
>
<!-- Manner of motion: repetition, size, manner, dynamics. -->

<!ENTITY % repetition
    "( fromstart
    | fromstart_several
    | tofroto
    | manyrandom
    | continue
    | continue_several
    | reverse
    | swap ) #IMPLIED"
>

```

```

<!ENTITY % size
  "( small
  | big ) #IMPLIED"
>

<!ENTITY % manner
  "( fast
  | slow
  | tense
  | rest
  | halt ) #IMPLIED"
>

<!ENTITY % incrdecr
  "( increasing | decreasing ) #IMPLIED"
>

<!--
dynamicsize_attribs is only used to describe how a wavy, zigzag, or
circular movement gets bigger or smaller.
-->

<!ENTITY % dynamicsize_attribs
  "incrdecr %incrdecr;
  incrdecr_size %size;"
>

<!ENTITY % clock_direction "( %the8directions; ) #IMPLIED">

<!ENTITY % zigzag_attribs
  "zigzag_style (zigzag | wavy) #IMPLIED
  zigzag_size %size;
  zigzag_incrdecr %incrdecr;
  zigzag_incrdecr_size %size;"
>

<!-- Hand Posture Attributes. -->

<!--
Explanation of the attributes of hand postures:

bodypart:
Usually a handconfig refers to a hand, but if it refers to
some other body part, such as the elbow, this is indicated by the
bodypart attribute.

handshape:
The basic HamNoSys handshape.

approx_shape:
True if the handshape is only approximate.

splay:
This defines the amount of splaying of the four fingers, from
0 (together) to 2 (maximally splayed). For physiological reasons,
splay has no effect on fingers with large amounts of bending at the
base joint. Thus the splay code effectively applies only to the
extended fingers. If splay is absent, the amount of splaying is
whatever is implied by the handshape. Splay values for individual
fingers may be overridden by later attributes implying such things

```

as contacts between fingers.

mainbend:

The general digit bending specification which may appear in a basic handshape in HamNoSys. It may be overridden by the individual digit bending specifications (see below). Note that for a pinch handshape this attribute specifies the kind of overlap between the thumb and the combining fingers. The value is either a string of numerical bend codes of the form "bbb", or one of a set of standard identifiers. The three bend codes apply to joints 1 (base of the digit), 2, and 3, respectively. Each bend value is in the range 0..4 (i.e. min..max), corresponding approximately to bend angles between 0 and 90 degrees).

Admissible named values, with the corresponding bend codes, are:

| | |
|-----------|-----|
| bent | 400 |
| round | 222 |
| hooked | 044 |
| dblent | 440 |
| dblhooked | 444 |
| halfent | 200 |

thumbpos:

This corresponds to the optional thumbposition in a basic HamNoSys handshape, apart from a cee handshape, (see ceeopening below). The value is one of a set of standard identifiers, or a pair of angle codes of the form "aa".

The allowed names correspond directly to the HamNoSys thumbposition, plus the value "halfout" (which recognizes a common application of the between operator). They are, with the corresponding angle codes:

| | |
|---------|----|
| out | 40 |
| halfout | 20 |
| across | 24 |
| opposed | 44 |

Note, however, that this type of thumbpos may imply bending attributes for thumb joints 2 and 3 (e.g. "across"), while the second type using angle codes does not. Each angle code is a digit in the range 0..4 (i.e. min..max). The first describes the angle between the thumb and the extended index finger: 0 for parallel and 4 for maximally extended. The second describes the rotation of the thumb about the axis of the extended index finger: 0 when the thumb is in the plane of the palm and 4 when it is maximally rotated in front of the palm. If thumbpos is omitted it implies the thumb is alongside the edge of the palm (equivalent to the bend code 00).

ceeopening:

This corresponds to the HamNoSys thumbposition in the case of a cee handshape.

specialfingers:

A string of single-digit finger identifiers. Depending on the handshape, it specifies a nonstandard set of extended finger(s), or a non-standard set of combining fingers in a thumb combination. (Corresponds to relevant instances of HamNoSys fingernothumb*.)

exempteddigits:

Similar format to specialfingers above, but it may also include the thumb identifier (1); the specified digits are exempted from the mainbend specification described above.

(Corresponds to HamNoSys fingernothumb* and the third HamNoSys thumbspecials option.)

bend1, bend2, bend3, bend4, bend5:

Explicit individual digit bending specifications, with the same permitted values as the mainbend attribute described above.

These specifications override those defined explicitly or implicitly by earlier attributes.

(These can be used to represent to HamNoSys fingershape* modifiers.)

contactpair, contactkind:

These jointly specify a HamNoSys fingercrossing;

contactpair, of the form "dd", identifies the two digits (finger or thumb) involved; contactkind identifies the site on the first digit at which the second makes contact.

second_contactpair, second_contactkind:

These allow a second HamNoSys fingercrossing to be specified.

(It is assumed that no more than two such specifications are needed, at least until non-human avatars are used.)

thumbbetween:

This has the form "dd", identifying a pair of fingers between which the thumb is placed. (Corresponds to the first of the HamNoSys thumbspecials options.)

thumbenclosed:

If true specifies that the thumb is enclosed by the fingers of its hand, in the context of handshape with some or all of its fingers in the fist (double-hooked) posture.

(Corresponds to the second HamNoSys thumbspecials option, i.e. plain hambetween.)

thumbcontact:

This indicates a nonstandard contact point for the thumb on those finger(s) determined by other parts of the handshape specification. (Corresponds to the fourth and final HamNoSys thumbspecials option.)

opposedfinger:

A single-digit finger identifier. In the case where several fingers ("specialfingers" above) participate in a thumb-combination handshape, this may be used to indicate which of these fingers is opposite the thumb, or contacting it.

(Corresponds to the finger with the "thumb opposed" flag set, as introduced in HamNoSys 4.)

extfidir:

The direction in which the fingers would point if they were fully extended.

palmor:

The orientation of the palm about the axis of the extended middle finger.

abs_extfidir, abs_palmor:

True if the left/right aspect of extfidir and palmor is literal.

By default, "right" means the dominant side of the body and "left" the non-dominant side.

rel_extfidir, rel_palmor:
True if respective extfidir and palmor should track the subsequent movement.

approx_extfidir, approx_palmor:
True if the extfidir or palmor need only be approximate.

NB Synthesis software can ignore the approx_... attributes.

NB The order of attributes in the above list is significant in that a later attribute overrides any relevant predecessor, i.e. synthesis software should process them in the order given.

NB For each class of basic handshape in HamNoSys only a subset of these attributes may be defined, e.g. thumbbetween only makes sense when thumbpos is "across". We leave the complete characterization of such context-dependent constraints to the HamNoSys 4 definition.

-->

```
<!ENTITY % handconfig_attribs
  "bodypart %site_bodyarm;
  side %side;

  handshape %basichandshape;
  approx_shape %boolfalse;

  splay ( 0 | 1 | 2 ) #IMPLIED

  mainbend CDATA #IMPLIED

  thumbpos CDATA #IMPLIED
  ceeopening %ceeopening;

  specialfingers CDATA #IMPLIED
  exempteddigits CDATA #IMPLIED

  bend1 CDATA #IMPLIED
  bend2 CDATA #IMPLIED
  bend3 CDATA #IMPLIED
  bend4 CDATA #IMPLIED
  bend5 CDATA #IMPLIED

  contactpair CDATA #IMPLIED
  contactkind ( %fingerpart; ) #IMPLIED

  second_contactpair CDATA #IMPLIED
  second_contactkind ( %fingerpart; ) #IMPLIED

  thumbbetween CDATA #IMPLIED
  thumbenclosed %boolfalse;
  thumbcontact ( %fingerpart; ) #IMPLIED
  opposedfinger CDATA #IMPLIED

  extfidir ( %the26directions; ) #IMPLIED
  abs_extfidir %boolfalse;
  rel_extfidir %boolfalse;
  approx_extfidir %boolfalse;

  palmor ( %the8directions; ) #IMPLIED
  abs_palmor %boolfalse;
```

```

    rel_palmor %boolfalse;
    approx_palmor %boolfalse;"
>

<!ENTITY % handconfiguration
    "handconfig | split_handconfig"
>

<!-- Locations. -->

<!--
A basic location is used to identify a single location.
(For a double-handed sign, this may resolve to two separate
locations.)

A split_location specifies a separate location for each hand.
One of these locations may be a position on the opposite hand, but not
both.

A handconstellation may specify the detailed location of the hands
with respect to each other, including proximity, and the general
location in signing space of the hands thus configured.
-->

<!ENTITY % location "location_bodyarm | location_hand | use_locname" >
<!ENTITY % location_both
    "%location; | split_location | handconstellation"
>

<!-- Hand Posture. -->

<!ENTITY % posture "( %handconfiguration; )*, ( %location_both; )?" >

<!-- Kinds of basic motion. -->

<!-- crossmotion is an obsolete feature of Hamnosys 2.0. -->

<!ENTITY % simplemotion
    "directedmotion |
    circularmotion |
    wristmotion |
    crossmotion |
    fingerplay |
    changeposture |
    nomotion"
>

<!ENTITY % motion_attribs
    "manner %manner;
    bouncing %boolfalse;
    fused %boolfalse;
    alternating %boolfalse;
    second_alternating %boolfalse;
    abs_motion %boolfalse;"
>

<!ENTITY % motion
    "%simplemotion;

```

```

    | par_motion
    | seq_motion
    | split_motion
    | rpt_motion
    | tgt_motion"
>

<!ENTITY % brushing_location "%location;" >

<!--
Two repetition attributes are provided. This is because the effect of
certain combinations of repetition operators is not the composition of
their separate effects. They should therefore be placed in the same
node, rather than in nested rpt_motion elements. For example, if both
repetition and second_repetition are "fromstart", the effect should be
to perform the motion three times, not four times. When they are
"reverse" and "fromstart", the effect is to perform first the motion,
then its reverse, then the original motion. The reversed motion is not
repeated.
HamNoSys 4: further enhancements to the set of repetition attributes
(but note we currently treat alternating repetition attributes
separately, including them with the general motion attributes).
-->

<!ENTITY % repetition_attrbs
    "repetition %repetition;
    second_repetition %repetition;
    repetition_incrdecr %incrdecr;
    repetition_baseshift ( %the26directions; ) #IMPLIED
    baseshift_incrdecr %incrdecr;"
>

<!-- ##### ELEMENT definitions ##### -->

<!--
A sign_manual is either a sequence of sign_manuals, or a basic sign
consisting of nonmanual configuration, hand configuration, hand
constellation, and motion.

The set of handconfigs is to be understood as being implicitly in
parallel, not sequential. Each handconfig or split_handconfig may
specify some components of the configuration, the overall initial
configuration being the union of these. This allows e.g. a single
location to be specified, together with separate hand shapes. It
is an error for multiple handconfigs to specify inconsistent data.

The handconstellation specifies a relationship between the two hands,
typically sites on the two hands which are to be in contact or
proximity.

The movements are assumed to be implicitly in sequence.
-->

<!ELEMENT sign_manual
    ( ( nonmanualconfig? , %posture; , ( %motion; )* )
    | ( sign_manual, sign_manual+ )
    )
>
<!ATTLIST sign_manual

```

```

    both_hands      %boolfalse;
    lr_symm         %boolfalse;
    ud_symm         %boolfalse;
    oi_symm         %boolfalse;
    outofphase     %boolfalse;
    realspace       %boolfalse;
>

<!--ELEMENT handconfig EMPTY>
<!--ATTLIST handconfig %handconfig_attribs;>

<!--ELEMENT split_handconfig ( handconfig, handconfig ) >

<!--
In location_hand:
The second group of attributes, if present, indicate betweenness.
True digits value indicates the set of fingers defining the site: if
in addition site itself is defined as a fingerpart then the site is
the indicated part of the specified fingers.
The location_hand subelement is for contact.
-->

<!--ELEMENT location_hand ( location_hand? ) >
<!--ATTLIST location_hand
    location          %site_hand;
    side              %side;
    digits            CDATA          #IMPLIED
    approx_location   %boolfalse;

    second_location   %site_hand;
    second_side       %side;
    second_digits     CDATA          #IMPLIED
    approx_second_location %boolfalse;

    contact           %contact_hand;

    %def_locname_attrib;
>

<!--
In location_bodyarm:
An undefined location means "in neutral space".
The location_hand subelement is for contact/distance.
-->

<!--ELEMENT location_bodyarm ( location_hand? ) >
<!--ATTLIST location_bodyarm
    location          %site_bodyarm;
    side              %side;
    approx_location   %boolfalse;

    second_location   %site_bodyarm;
    second_side       %side;
    approx_second_location %boolfalse;

    behind           %boolfalse;
    contact           %contact_bodyarm;

    %def_locname_attrib;
>

```

```

<!ELEMENT use_locname EMPTY >
<!ATTLIST use_locname
    %use_locname_attrib;
>

<!ELEMENT split_location ( (%location;) , (%location;) ) >

<!--
In a handconstellation:
if location_hands are present, they are, in order:
    [of-dom-on-nondom, of-nondom-on-dom];
the optional location_bodyarm subelement (should really be restricted to
location_body only?) specifies the location of the handconstellation;
the contact attribute defines the distance relation between the hands.
-->

<!ELEMENT handconstellation (
    (location_hand, location_hand)?, location_bodyarm?
)>
<!ATTLIST handconstellation
    contact %contact_hand;
>

<!--
The DTD places no restrictions on how the three types of complex
motion may be nested inside each other. Given that in Hamnosys, there
are examples of any type of complex motion occurring inside any other,
there is no way within the DTD to exclude arbitrarily deep nesting
without artificially multiplying entities. We therefore let the DTD
allow all nestings, but may require restrictions on nesting for correct
SigML.

In the Hamnosys corpus, we have observed only the following nestings:
    par(seq)
    seq(par)
    par(split)
    split(par)
    seq(split)
    split(seq)
    split(par(seq))
seq(seq) may be necessary in order to express complex patterns of
repetition. Any one-handed motion should be allowed to occur within a
split motion. We propose that only these nestings be allowed. It is
up to each SigML processor to decide what complexity of nesting it will
correctly handle. Excessive nesting should never cause a program error,
but result only in some components of the motion being ignored. Further
experience in the processing of SigML will reveal whether arbitrary
nesting causes any significant complications.
-->

<!ELEMENT split_motion ( ( %motion; ) , ( %motion; ) ) >
<!ATTLIST split_motion %motion_attribs;>

<!ELEMENT par_motion ( ( %motion; ) , ( %motion; )+ ) >
<!ATTLIST par_motion %motion_attribs;>

<!ELEMENT seq_motion ( ( %motion; ) , ( %motion; )+ ) >
<!ATTLIST seq_motion %motion_attribs;>

<!ELEMENT tgt_motion ( ( %motion; ) , ( %posture; ) ) >

```

```

<!ATTLIST tgt_motion %motion_attribs;>

<!ELEMENT rpt_motion ( %motion; ) >
<!ATTLIST rpt_motion
    %motion_attribs;
    %repetition_attribs;
>

<!--
nonmanualconfig specifies an initial configuration of non-manual
elements. It is rarely used. We allow an arbitrary set of handconfigs
here, which is far too liberal, but certainly includes everything that
can validly appear. It may include motions, but a motion in this
context should not be understood as a real motion, but as a
specification of an intended position relative to the neutral position.
For example, raised shoulders would be notated as a small upward
movement of the shoulders.
-->

<!ELEMENT nonmanualconfig ( handconfig* )>

<!-- Motion types -->

<!--
In some cases, a simple motion element may have a brushing contact
location as a sub-element.
(RE, 2001-08)
-->

<!--
When the "second" attributes of directedmotion are present, the
meaning is betweenness. This is rarely used.

The ellipse_direction refers to the plane in which the zigzag motion
happens.
-->

<!ELEMENT directedmotion ( (%brushing_location;)? ) >
<!ATTLIST directedmotion
    direction (%the26directions;) #REQUIRED
    size %size;
    second_direction (%the26directions;) #IMPLIED
    second_size %size;

    curve (%the8directions;) #IMPLIED
    curve_size %size;

    %zigzag_attribs;
    ellipse_direction %ellipse_direction;

    %motion_attribs;

    %def_locname_attrib;
>

<!--
The dynamicsize_attribs refers to the dynamics of the circular
motion.

```

If a zigzag is present, it is assumed to be of constant size and perpendicular to the plane of the circle.

-->

```
<!ELEMENT circularmotion ( (%brushing_location;)? ) >
<!ATTLIST circularmotion
  axis (%the26directions;) #REQUIRED
  second_axis (%the26directions;) #IMPLIED
  size %size;

  start %clock_direction;
  clockplus %boolfalse;
  second_clockplus %boolfalse;
  end %clock_direction;

  ellipse_direction %ellipse_direction;
  ellipse_size %size;

  %zigzag_attribs;

  %dynamicssize_attribs;
  %motion_attribs;

  %def_locname_attrib;
>
```

```
<!ELEMENT wristmotion EMPTY>
<!ATTLIST wristmotion
  motion ( %wristmotion; ) #REQUIRED
  size %size;

  %motion_attribs;
>
```

<!-- crossmotion is an obsolete(?) feature of Hamnosys 2.0. -->

```
<!ELEMENT crossmotion EMPTY>
<!ATTLIST crossmotion
  cross ( plus | x ) #REQUIRED
  %motion_attribs;
>
```

```
<!ELEMENT fingerplay EMPTY>
<!ATTLIST fingerplay
  %motion_attribs;
>
```

<!--
change posture should only occur in a tgt_motion, when the required
motion consists only of the change of hand configuration specified
by the associated posture.
-->

```
<!ELEMENT changeposture EMPTY>
<!ATTLIST changeposture
  %def_locname_attrib;
>
```

```
<!--  
nomotion is used as a placeholder when a split_motion specifies  
motion for only one hand.  
-->
```

```
<!ELEMENT nomotion EMPTY>
```

```
<!-- End of SiGML Manual DTD -->
```

DTD for Non-Manual SiGML

```
<!-- sigmlnonmanual.dtd

This is the DTD for the non-manual subset of core SiGML 1.0,
defining the sign_nonmanual element.
-->

<!--File Name:      SigmlCNonManual.dtd          -->
<!--Author:        Kevin Parsons                -->
<!--Description:   Definition of non-manual extension to      -->
<!--              HamNoSys as described in 'ViSiCAST        -->
<!--              Deliverable D5-1: Interface Definitions'   -->
<!--              (D5-1v012.pdf)                       -->
<!--Date Created:  27 March 2001                 -->
<!--Date Modified: 18 May 2001                  -->
<!--Date Modified: 29 May 2001                  -->
<!--Date Modified: 2001-12-08 R.E.              -->
<!--              - Added entities for abbreviated tags.    -->

<!-- Entities -->

<!ENTITY % shoulder_movement_abbrev
  "UL | UR | UB | HL | HR | HB | SL | SR | SB"
>

<!ENTITY % shoulder_movement
  "%shoulder_movement_abbrev;
  UL_left_shoulder_raised
  UR_right_shoulder_raised
  UB_both_shoulders_raised
  HL_left_shoulder_hunched_forward
  HR_right_shoulder_hunched_forward
  HB_both_shoulders_hunched_forward
  SL_left_shoulder_shrugging_up_and_down
  SR_right_shoulder_shrugging_up_and_down
  SB_both_shoulders_shrugging_up_and_down
  "
>

<!ENTITY % body_movement_abbrev
  "RL | RR | TL | TR | TF | TB | SI | HE | ST | RD"
>

<!ENTITY % body_movement
  "%body_movement_abbrev;
  RL_rotated_left
  RR_rotated_right
  TL_tilted_left
  TR_tilted_right
  TF_tilted_forwards
  TB_tilted_backwards
  SI_sigh
  HE_heave
  ST_straight
  RD_round
  "
>

<!ENTITY % head_movement_abbrev
  "NO | SH | SR | SL | TR | TL | NF | NB | PF | PB | LI"
```

```

>
<!ENTITY % head_movement
    "%head_movement_abbrev;
    NO_nodding_up_and_down
    SH_shaking_left_and_right
    SR_turned_right
    SL_turned_left
    TR_tilted_right
    TL_tilted_left
    NF_tilted_forward
    NB_tilted_back
    PF_pushed_forward
    PB_pushed_backward
    LI_head_movement_linked_to_eye_gaze
    "
>

<!ENTITY % eye_gaze_abbrev
    "AD | FR | HD | HI | HC | UP | DN | LE | RI | NO | RO"
>

<!ENTITY % eye_gaze
    "%eye_gaze_abbrev;
    AD_towards_addressee
    FR_far
    HD_towards_the_signer_s_own_hands
    HI_towards_the_signer_s_own_dominant_hand
    HC_towards_the_signer_s_own_non_dominant_hand
    UP_up
    DN_down
    LE_left
    RI_right
    NO_no_target_unfocussed
    RO_rolling_eyes
    "
>

<!ENTITY % eye_brows_abbrev
    "RB | RR | RL | FU"
>

<!ENTITY % eye_brows
    "%eye_brows_abbrev;
    RB_both_eyebrows_raised
    RR_right_eyebrow_raised
    RL_left_eyebrow_raised
    FU_eye_brows_furrowed
    "
>

<!ENTITY % eye_lids_abbrev
    "WB | WR | WL | SB | SR | SL | CB | CR | CL | TB | TR | TL | BB"
>

<!ENTITY % eye_lids
    "%eye_lids_abbrev;
    WB_wide_open_eyelids
    WR_wide_open_right_eyelid
    WL_wide_open_left_eyelid
    SB_narrowed_almost_closed_eyelids_slits
    "

```

```

SR_narrowed_almost_closed_right_eyelid
SL_narrowed_almost_closed_left_eyelid
CB_closed_eyelids
CR_closed_right_eyelid
CL_closed_left_eyelid
TB_tightly_shut_eyelids
TR_tightly_shut_right_eyelid
TL_tightly_shut_left_eyelid
BB_eye_blink_at_the_very_end_of_a_sign
"
>
<!ENTITY % nose_abbrev
  "WR | TW | WI"
>
<!ENTITY % nose
  "%nose_abbrev;
  WR_wrinkled_nose
  TW_twitching_nose
  WI_widened_nostrils
  "
>
<!ENTITY % mouth_gesture_abbrev
  " D01 | D02 | D03 | D04 | D05 | D06 | D07 | J01
  | L01 | L02 | L03 | L04 | L05 | L06 | L07 | L08 | L09 | L10
  | L11 | L12 | L13 | L14 | L15 | L16 | L17 | L18 | L19 | L20
  | L21 | L22 | L23 | L24 | L25 | L26
  | C01 | C02 | C03 | C04 | C05 | C06 | C07 | C08 | C09 | C10
  | T01 | T02 | T03 | T04 | T05 | T06 | T07 | T08 | T09 | T10
  | T11 | T12 | T13 | T14 | T15"
>
<!ENTITY % mouth_gesture
  "%mouth_gesture_abbrev;
  D01_eee_sss
  D02_f
  D03_ef
  D04_af
  D05_clattering_teeth
  D06_clattering_teeth_with_raised_upper_lip
  D07_one_bite_resulting_in_closed_teeth
  J01_lower_jaw_moves_sideways_left_and_right
  L01_sh
  L02_prrr
  L03_pr
  L04_pursed_lips
  L05_o_oa_open_o
  L06_ooo_closed_o
  L07_oa
  L08_boam
  L09_bam
  L10_boa
  L11_ba
  L12_bee
  L13_pi
  L14_pch
  L15_bsss_bee
  L16_pf
  L17_p

```

```

L18_p_p_p
L19_phh
L20_phh
L21_ph
L22_ph
L23_mmm
L24_mmm_while_holding_breath
L25_m_m_m
L26_one_side_of_upper_lip_raised
C01_puffed_cheeks
C02_cheeks_and_lip_area_puffed
C03_gradually_puffing_cheeks
C04_one_cheek_puffed
C05_one_cheek_puffed_while_briefly_blowing_out_air
C06_one_cheek_puffed_briefly_blowing_air_cheek_pushed
C07_cheeks_sucked_in
C08_cheeks_sucked_in_sucking_in_air
C09_tongue_pushed_visibly_into_cheek
C10_tongue_repeatedly_pushes_into_cheek
T01_l
T02_tip_of_tongue_slightly_protruding
T03_l_l_l
T04_tongue_sticks_out_briefly
T05_a
T06_tongue_sticking_out_repeatedly
T07_lalala
T08_alalal
T09_als
T10_lf
T11_laf
T12_tip_of_tongue_touches_one_corner_of_the_mouth
T13_tongue_tip_between_lower_lip_lower_teeth_middle_tongue_showing |
T14_tip_of_tongue_is_protruded_and_moving_sidewards |
T15_oval_circling_movement_of_tongue_in_open_mouth"
>

<!ENTITY % mouthmetatype
"mouthmetatype (holdprevious|stretchprevious) #REQUIRED"
>

<!ENTITY % nonmanual_start_timing
"slight_delay |
start_slightly_ahead
"
>

<!ENTITY % nonmanual_end_timing
"lasts_longer |
ends_before
"
>

<!ENTITY % shoulder_item
"( shoulder_movement | shoulder_par)"
>

<!ENTITY % body_item
"( body_movement | body_par)"
>

<!ENTITY % head_item

```

```

    "( head_movement | head_par)"
>
<!ENTITY % eye_gaze_item
    "( eye_gaze | eye_par)"
>
<!ENTITY % facial_expression
    "(eye_brows | eye_lids | nose)"
>
<!ENTITY % facial_expression_item
    "( %facial_expression; | facial_expr_par)"
>
<!ENTITY % mouthing
    "( mouth_gesture | mouth_picture | mouth_meta)"
>
<!ENTITY % mouthing_item
    "( %mouthing; | mouthing_par)"
>
<!ENTITY % synchronization
    "presynchronization      (%nonmanual_start_timing;) #IMPLIED
    postsynchronization      (%nonmanual_end_timing;) #IMPLIED"
>
<!-- Top level element -->
<!ELEMENT sign_nonmanual
    (
        shoulder_tier?,
        body_tier?,
        head_tier?,
        eyegaze_tier?,
        facialexpr_tier?,
        mouthing_tier?
    )
>
<!-- Tier elements -->
<!ELEMENT shoulder_tier
    ( %shoulder_item;, ( neutral?, %shoulder_item;)* )
>
<!ATTLIST shoulder_tier
    %synchronization;
>
<!ELEMENT body_tier
    ( %body_item;, ( neutral?, %body_item;)* )
>
<!ATTLIST body_tier
    %synchronization;
>
<!ELEMENT head_tier
    ( %head_item;, ( neutral?, %head_item;)* )
>
<!ATTLIST head_tier

```

```

    %synchronization;
>

<!ELEMENT eyegaze_tier
  ( %eye_gaze_item;, ( neutral?, %eye_gaze_item;)* )
>
<!ATTLIST eyegaze_tier
  %synchronization;
>

<!ELEMENT facialexpr_tier
  ( %facial_expression_item;, ( neutral?, %facial_expression_item;)* )
>
<!ATTLIST facialexpr_tier
  %synchronization;
>

<!ELEMENT mouthing_tier
  ( %mouthing_item;, ( neutral?, %mouthing_item;)* )
>
<!ATTLIST mouthing_tier
  %synchronization;
  fitpictureto manual (true | false) "false"
>

<!-- Parallel items -->
<!ELEMENT shoulder_par
  ( shoulder_movement, shoulder_movement+ )
>

<!ELEMENT body_par
  ( body_movement, body_movement+ )
>

<!ELEMENT head_par
  ( head_movement, head_movement+ )
>

<!ELEMENT eye_par
  ( eye_gaze, eye_gaze+ )
>

<!ELEMENT facial_expr_par
  ( %facial_expression;, (%facial_expression;)+ )
>

<!ELEMENT mouthing_par ( %mouthing;, (%mouthing;)+ )
>

<!-- Movement elements -->

<!ELEMENT shoulder_movement EMPTY>
<!ATTLIST shoulder_movement
  movement (%shoulder_movement;) #REQUIRED
>

<!ELEMENT body_movement EMPTY>
<!ATTLIST body_movement
  movement (%body_movement;) #REQUIRED
>

```

```

<!ELEMENT head_movement EMPTY>
<!ATTLIST head_movement
    movement (%head_movement;) #REQUIRED
>

<!ELEMENT eye_gaze EMPTY>
<!ATTLIST eye_gaze
    direction (%eye_gaze;) #REQUIRED
>

<!ELEMENT eye_brows EMPTY>
<!ATTLIST eye_brows
    movement (%eye_brows;) #REQUIRED
>

<!ELEMENT eye_lids EMPTY>
<!ATTLIST eye_lids
    movement (%eye_lids;) #REQUIRED
>

<!ELEMENT nose EMPTY>
<!ATTLIST nose
    movement (%nose;) #REQUIRED
>

<!ELEMENT mouth_gesture EMPTY>
<!ATTLIST mouth_gesture
    movement (%mouth_gesture;) #REQUIRED
>

<!ELEMENT mouth_picture EMPTY>
<!ATTLIST mouth_picture
    picture CDATA #REQUIRED
>

<!ELEMENT mouth_meta EMPTY>
<!ATTLIST mouth_meta
    %mouthmetatype;
>

<!ELEMENT neutral EMPTY>

```

(this page intentionally left blank)

Appendix B (D4-2)

Status of AnimGen's Implementation of SiGML Features

The following is a raw summary of SiGML features not yet implemented (2002-01-04). *No features of non-manual SiGML* are yet implemented with the exception of eye-brow raising and furrowing.

The remaining unimplemented features are of three types:

- unimplemented elements;
- unimplemented attributes of implemented elements;
- unimplemented combinations of implemented elements.

Unimplemented Elements

N.B. As stated above, all non-manual elements are unimplemented, except those facial-tier elements for eye-brow raising and furrowing.

High-Level Elements:

avatar hamgestural_segment bonesanimation_sign mocap_sign

Manual Elements:

wristmotion fingerplay nonmanualconfig

Unimplemented Attributes of Implemented Elements

<location_bodyarm>:
 approx_second_location behind approx_location

<gloss_sign>:
 speed

<hamgestural_sign>:
 speed

<rpt_motion>:
 fused bouncing brushing manner abs_motion

<split_motion>:
 fused bouncing brushing manner abs_motion

<sign_manual>:
 realspace outofphase

<tgt_motion>:
 fused bouncing brushing manner abs_motion

<circularmotion>:

bouncing brushing incrdecr_size manner incrdecr
fused zigzag_incrdecr_size zigzag_size
zigzag_incrdecr zigzag_style
abs_motion

<par_motion>:
fused bouncing brushing manner abs_motion

<handconfig>:
rel_extfidir bodypart use_locname thumbenclosed def_locname
rel_palmor approx_palmor thumbcontact ceopening
approx_extfidir
splay abs_extfidir abs_palmor thumbbetween second_contactkind
second_contactpair contactkind contactpair

<directedmotion>:
bouncing zigzag_incrdecr_size brushing manner zigzag_size
fused zigzag_style zigzag_incrdecr abs_motion

<facialexpr_tier>:
postsynchronization presynchronization

<location_hand>:
approx_second_location approx_location

<seq_motion>:
fused bouncing brushing manner abs_motion

Unimplemented Combinations of Implemented Elements

<rpt_motion> has unimplemented subelements:
<seq_motion>

<sign_manual> has unimplemented subelements:
<sign_manual>

<tgt_motion> has unimplemented subelements:
<seq_motion>

<split_motion> has unimplemented subelements:
<seq_motion>

<par_motion> has unimplemented subelements:
<seq_motion>

Appendix C (D4-2)

***Animgen* VRML GUI Widgets**

Dial

This rotates once per cycle of the animation sequence; click or drag on the dial face to control the animation manually. The gloss-name of the current sign is displayed below the dial.

Right/Left Arrowhead

Indicator/toggle control for the temporal direction of the animation; the default is "to the future" (right arrow); the alternative is "to the past" (left arrow).

Square

Play/Pause the animation.

Up/Down Arrowheads, Spot

These control animation speed; the "up" arrow increases the animation speed by 41% (so clicking it twice doubles the speed); the "down" arrow reduces speed by 41%; the "spot" resets the speed to normal.

Perforated Scroll Wheel (to the left of the speed controls)

This can be dragged to change the position in the animation sequence.

Plus/Minus buttons

These allow single-stepping in either direction through the frames of the animation sequence.

Mauve Diamonds

These toggle the display/hiding of "significant sites" related to the body (left diamond), and to the hands and face (right diamond), of the avatar.

(this page intentionally left blank)

HamNoSys to Animation Components

Introduction

This document describes the implementation of the BAF Player component (BAFPlayer3.dll) and the SiGMLToAnimation component (SiGMLToAnimCtrls.ocx) that are used in the translation of HamNoSys to Animation. It also describes how both are embedded in a web page and a Visual Basic 6 application and used to translate and play synthesized signs.

BAF Player Component

Summary

The BAF Player is implemented as a COM component in BAFPlayer3.dll. It is written in Visual C++ using the Active Template Library. The main functionality of the BAFPlayer is implemented in BAFPlayer.cpp, and defined in BAFPlayer.h.

It contains the Televirtual IHostCom Avatar component with which it interacts, by initializing the Avatar, setting up the Avatar's lights and cameras, and resizing the Avatar.

It also implements methods that allow the Playing, Pausing, Stepping and Stopping of a BAF, XBAF or SiGML file, and it also fires events that allow a client container to know which file is playing, what frame is playing and so on.

When the 'Play' method is called, this in turn calls the 'StartPlaying' function that carries out the following:

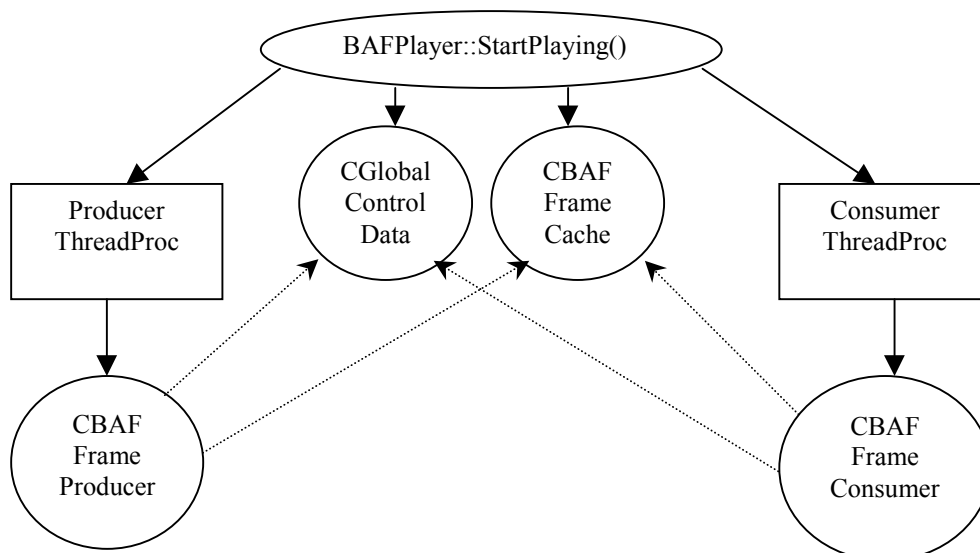
- Creates an instance of a CGlobalControlData class
- Create an instance of a CBAFFrameCache class
- Creates a Consumer Thread
- Creates a Producer Thread

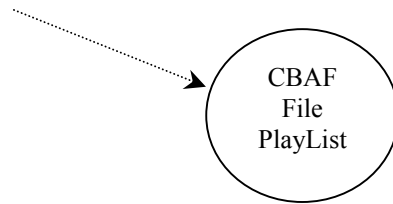
All of these are discussed in detail below.

Playing a BAF or XBAF File

The top-level software objects that are created and used while playing a BAF or XBAF File are depicted in the following diagram, where:

- An ellipse is a function call
- A rectangle is a function called in a separate thread
- A circle is a class
- A solid-lined arrow means 'creates'
- A dashed-line arrow means 'uses'





CGlobalControlData

This class is implemented in GlobalControlData.cpp, and defined in GlobalControlData.h.

This class centralizes the control of the Consumer and Producer threads and control of access to the Frame Cache. It uses Semaphores to suspend the execution of a thread, e.g. the function SetProducerWaiting(). It uses Critical Sections to control access to each of the global data items, e.g. the functions GetProducerWaiting() and LockCache().

CBAFFrameCache

This class is implemented in BAFFrameCache.cpp, and defined in BAFFrameCache.h.

This class uses the STL class 'queue' as a buffer into which BAFFrames can be inserted and removed.

CBAFFilePlayList

This class is implemented in BAFFilePlayList.cpp, and defined in BAFFilePlayList.h.

This class uses the STL class 'list' as a play-list of the files to play.

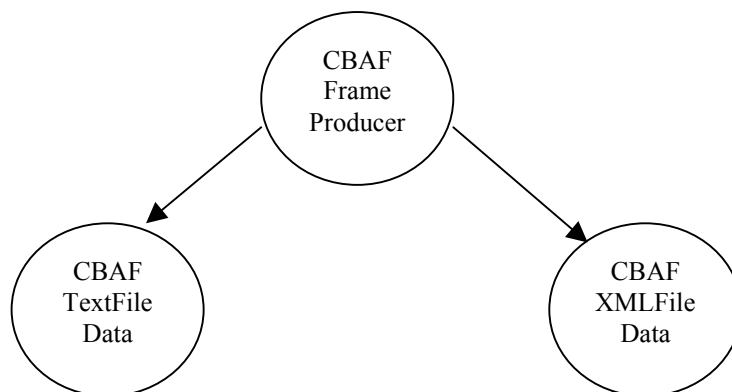
CBAFFrameConsumer

This class is implemented in 'BAFFrameConsumer.cpp' and defined in 'BAFFrameConsumer.h'.

This class retrieves frames from the frame cache and passes them on to the Televirtual Avatar component using the 'SetBones' method. This class has three member functions, the only public one being 'ReadFramesFromCache'.

CBAFFrameProducer

This class is implemented in 'BAFFrameProducer.cpp' and defined in 'BAFFrameProducer.h'.



This class inserts frames into the frame cache after getting a frame from either a CBAFTextFileData or CBAFXMLFileData class, which are described below. This class has only two public member functions, 'Init' and 'ReadFramesIntoCache'.

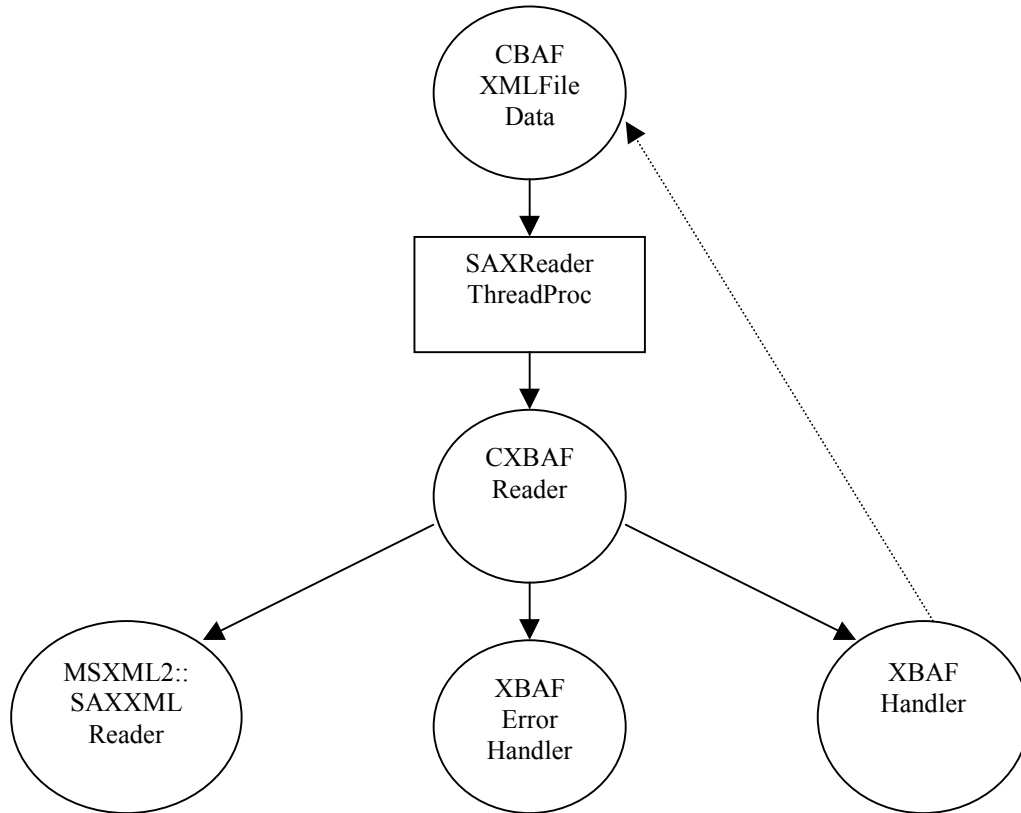
CBAFTextFileData

This class is implemented in 'BAFTextFileData.cpp' and defined in 'BAFTextFileData.h'. It is derived from the base class 'CBAFFileData' which is defined in 'BAFFileData.h'.

This class extracts data from a BAF file.

CBAFXMLFileData

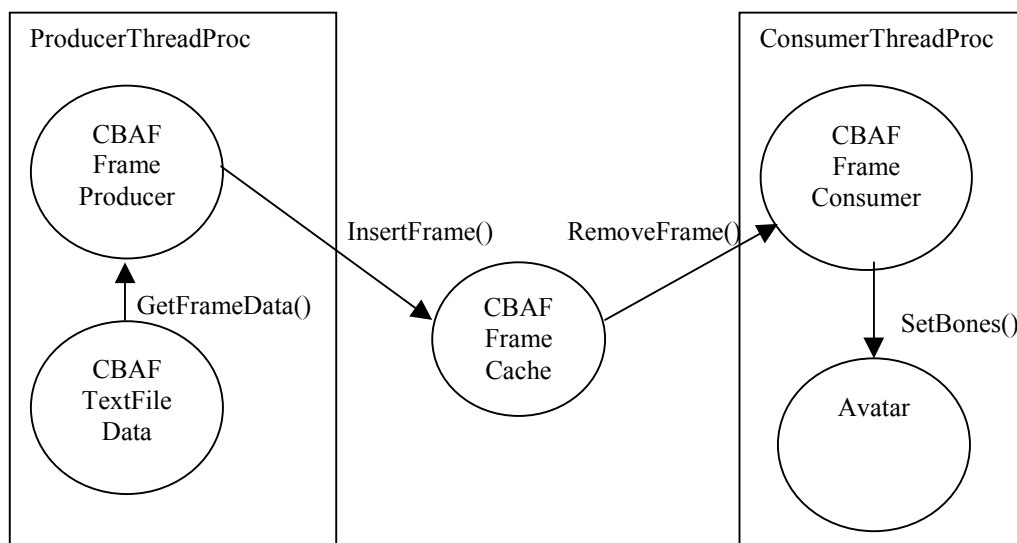
This class is implemented in 'BAFXMLFileData.cpp' and defined in 'BAFXMLFileData.h'. It is derived from the base class 'CBAFFileData' which is defined in 'BAFFileData.h'.



This class uses a SAXXMLReader to extract data from an XBAF file. It uses an STL queue to store frames that have been passed to it by the XBAFHandler class. A frame is removed from this queue and passed to the CBAFFrameProducer class when the 'GetFrameData' function is called.

BAF Player Frame Data Flow

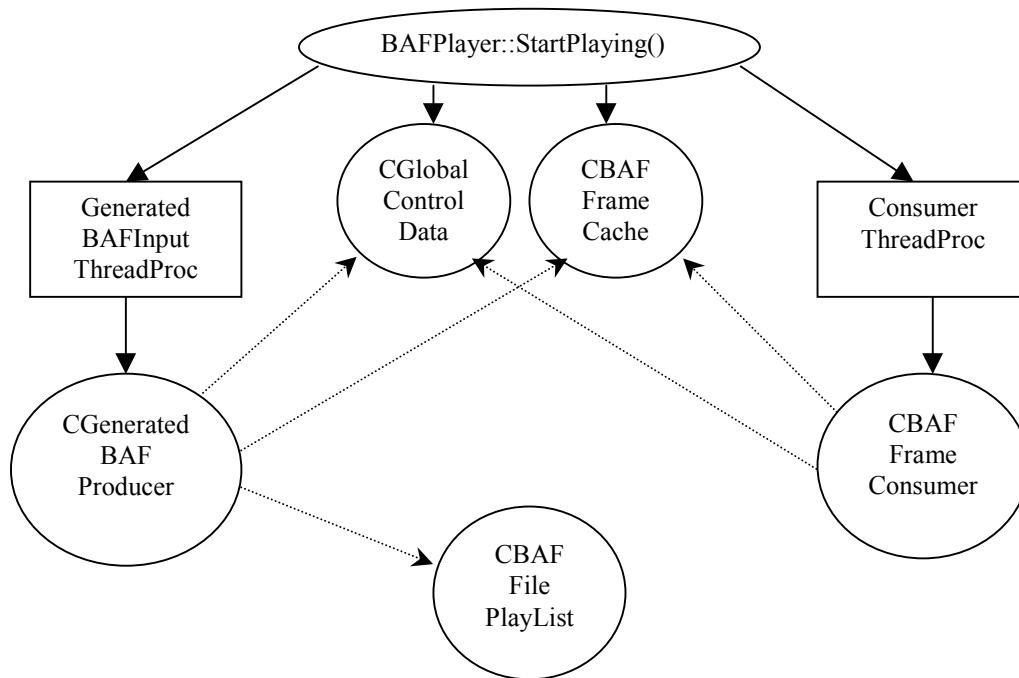
The flow of frame data when the BAF Player is running is depicted in the following diagram:



The Consumer Thread is created in a suspended state, and is activated by the Producer thread when the Frame Cache contains a certain number of frames. This can be set as a percentage of the total frames of the first item in the play-list. If no percentage is given then the default is 50%. So, using the default of 50%, if a BAF file contained 30 frames, the BAF Consumer thread would be inactive until the sixteenth frame was inserted into the Frame Cache. It would then start retrieving frames from the Frame Cache and passing them to the Avatar using the 'SetBones' method.

Playing a Generated BAF

The top-level software objects that are created when playing a BAF generated from SiGML are depicted in the following diagram:

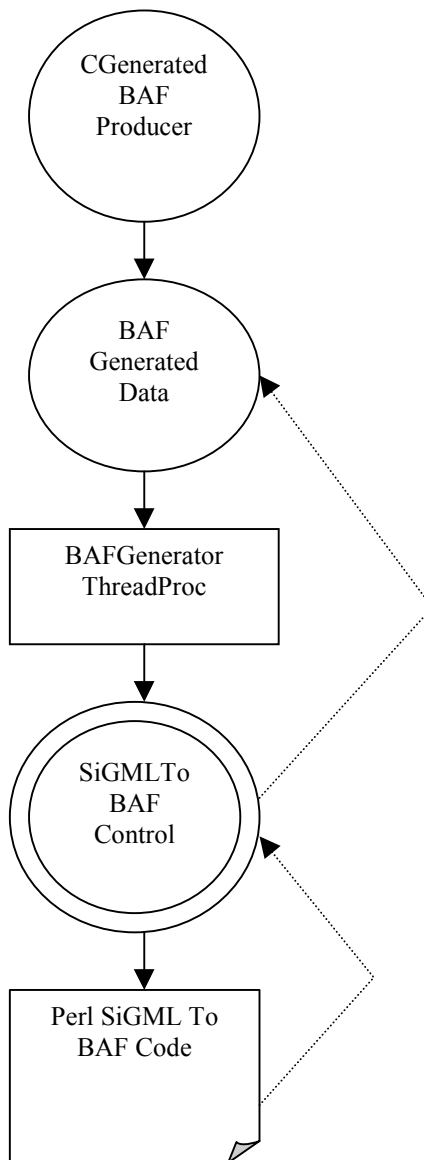


In this case the CBAFFPlayList contains SiGML files.

CGeneratedBAFProducer

This class is implemented in 'CGeneratedBAFProducer.cpp' and defined in 'CGeneratedBAFProducer.h'.

This class is very similar to the CBAFFrameProducer class documented above, except that instead of creating sub-ordinate C++ classes to do the work it creates COM objects.

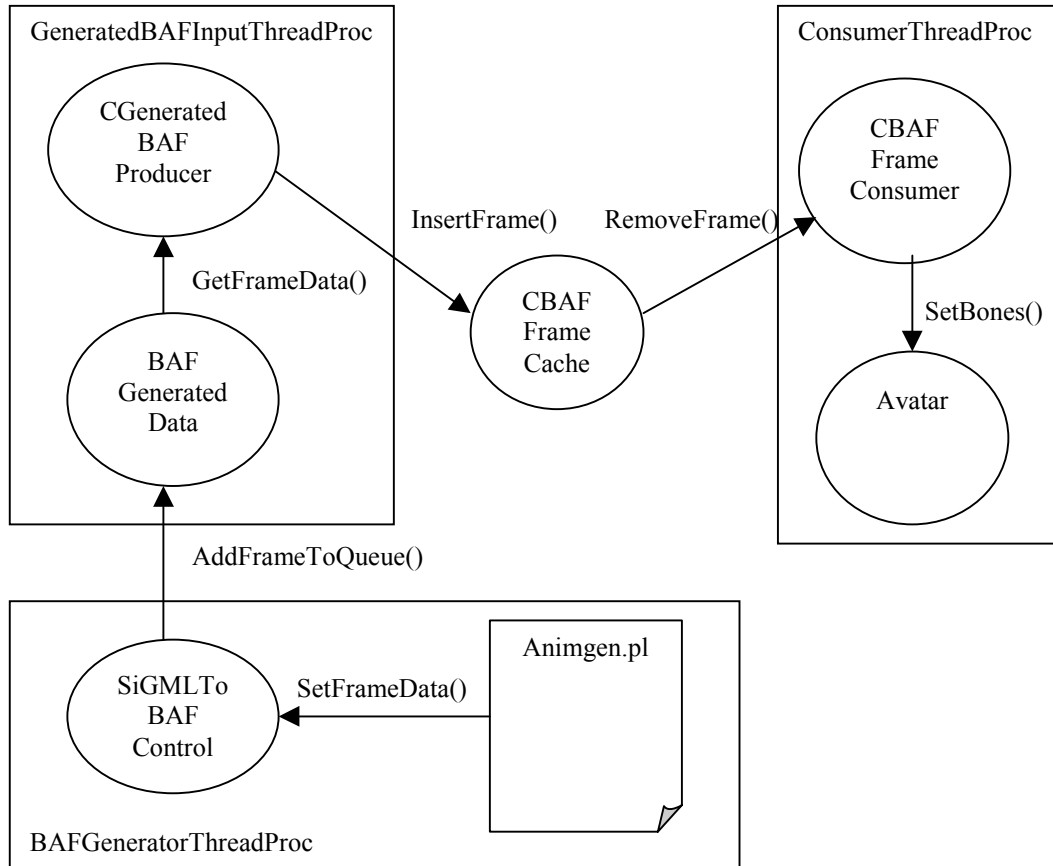


BAFGeneratedData

This component is implemented in 'BAFGeneratedData1.cpp' and defined in 'BAFGeneratedData1.h'.

This component uses an STL queue to store the frames that are passed to it by the SiGMLToBAF control (see below), using the 'AddFrameToQueue' method. A Frame is retrieved from this queue when the CGeneratedBAFProducer class calls the 'GetFrameData' method.

BAF Player Generated Frame Data Flow



Stopping the BAF Player

In normal circumstances when each part of the hierarchy of software objects has finished its task it terminates. So in the Generated BAF example above, the software objects terminate in this order:

1. Animgen.pl
2. SiGMLToBAF Control
3. BAFGeneratorThreadProc
4. BAFGeneratedData
5. CGeneratedBAFProducer
6. GeneratedBAFInputThreadProc
7. CBAFFrameConsumer
8. ConsumerThreadProc

This order, not surprisingly, is the reverse of the order in which the objects are created. The ConsumerThreadProc sends a message to the BAF Player once it has finished playing, and is about to terminate. When this message is received, the BAF Player deletes the CGlobalControlData and CBAFFrameCache objects.

If the BAF Player is stopped before all frames have been played, usually by the client calling the 'Stop' method, then the 'ProducerAborted' and 'ConsumerAborted' flags are set to True.

The 'ProducerAborted' flag is checked by the ReadFramesIntoCache function of the CBAFFrameProducer class for each iteration of the while loop within the function. The value of this flag is passed down to all software objects used by the producer hierarchy of the BAF Player. So, in the generated example, the Perl code, Animgen.pl, is the last to receive the value of this flag. If it finds that the BAF producing process has been stopped it stops generating BAF frames and terminates. This causes its creator, the SiGMLToBAF Control, to terminate, and so on up the hierarchy. So, terminating all the software objects when stopping is exactly the same as the finishing scenario described above.

The 'Stop' method also fires a 'Stop' event, as there are circumstances when the Producer thread is suspended waiting for input. The receipt of the 'Stop' event unblocks the thread allowing it to terminate, as described above.

The 'ConsumerAborted' flag is checked by the ReadFramesFromCache function of the CBAFFrameConsumer class for each iteration of the while loop within the function. If the 'ConsumerAborted' flag is True then the while loop terminates, the ReadFramesFromCache terminates, the CBAFFrameConsumer class terminates and then finally, the ConsumerThreadProc thread terminates.

SIGMLToAnimation Component

This component is implemented in the 'SiGMLToAnimCtrls.ocx' file. It is written in Visual C++ using MFC. It contains two controls, the SiGMLToBAF Control and the SiGMLToVRML Control. Both of these controls are described below.

Both controls run code written in Perl to carry out the appropriate translation. The method use to call Perl code from C++ is as described in the 'perlembd' section of the ActivePerl documentation.

The registry stores the path and the filename of the Perl code to be executed under the 'HKEY_LOCAL_MACHINE\SOFTWARE\ViSiCAST\SIGMLToAnimation' key.

SIGMLToBAF Control

This is implemented as a separate control in the 'SiGMLToAnimCtrls.ocx' file. It is implemented in 'SiGMLToBAFctl.cpp' and defined in 'SiGMLToBAFctl.h'.

This control causes the Perl code that translates SiGML to BAF to be run, by creating an appropriate command line with the options that tell the Perl code to output a BAF file.

This control registers itself in the Running Object Table (ROT). When Generated BAFs are required this allows the Perl code to get a reference to this control and hence call methods on it. In this manner, data, such as a total frame count and frame data itself, is passed back to the control, which in turn allows it to pass this data back up the hierarchy as described in the section 'BAF Player Generated Frame Data Flow'.

SIGMLToVRML Control

This is implemented as a separate control in the 'SiGMLToAnimCtrls.ocx' file. It is implemented in 'SiGMLToVRMLctl.cpp' and defined in 'SiGMLToVRMLctl.h'.

This control causes the Perl code that translates SiGML to VRML to be run, by creating an appropriate command line with the options that tell the Perl code to output a VRML file.

Perl Code

Additions To Perl Code

To make the Perl code able to talk to the SiGMLToBAF control the Perl module 'Communicate::CallerObject' was implemented. This uses the 'GetActiveObject' method to attach to the control in the ROT. It also provides Perl sub-routines that wrap the functionality provided by the control.

Additional code was also added to the main Perl file, 'animgen.pl', all of these additions are have the following before it: '# !! Added by KJP - 2001-11-20' and '# !!' after it.

The unit of transfer from the Perl module to the SiGMLToBAF control is a BAF Frame. The 'SetFrameData' method provided by the SiGMLToBAF control requires three parameters. The first is of type float and denotes the time-stamp of the frame. The second is of type SAFEARRAY of float and contains the data that constitutes a BAF frame. The third is of type integer and denotes the index of the frame, i.e. its position with regard to all the frames in the BAF file, or of all the frames to be generated.

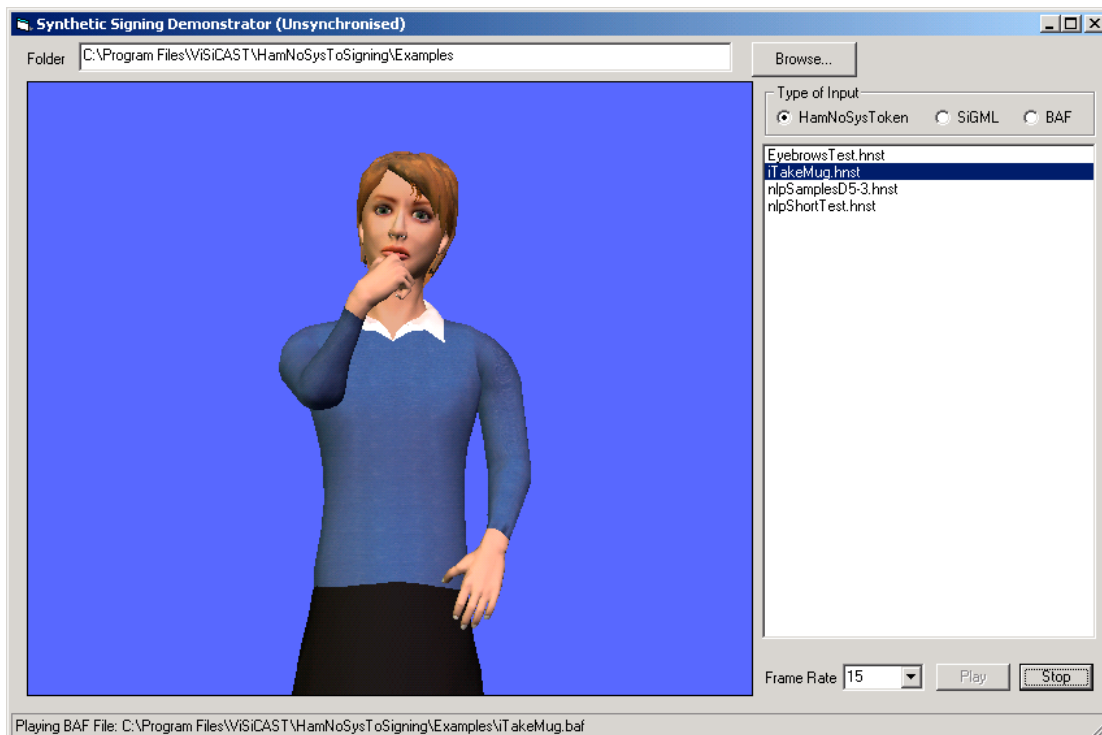
Translation of HamNoSys To SiGML

If the 'UseExternalSiGMLGenerator' property is set to TRUE in the SiGMLToBAF control, then another additional Perl module, 'HamNoSysToSiGMLGenerator.pm' is used to create and interact with the 'H2AFECtrl' control which is a COM control of the HamNoSys to SiGML Java converter.

The unit of transfer between the 'H2AFECtrl' control and the Perl module is a SiGML sign, i.e. XML document fragments that start <hamgestural_sign> and end </hamgestural_sign>.

BAF Player Front End

To allow a user to use the software described above, a Visual Basic 6 application was created that contained the above controls and provided a user interface to interact with them. A screen shot of this application is given below:

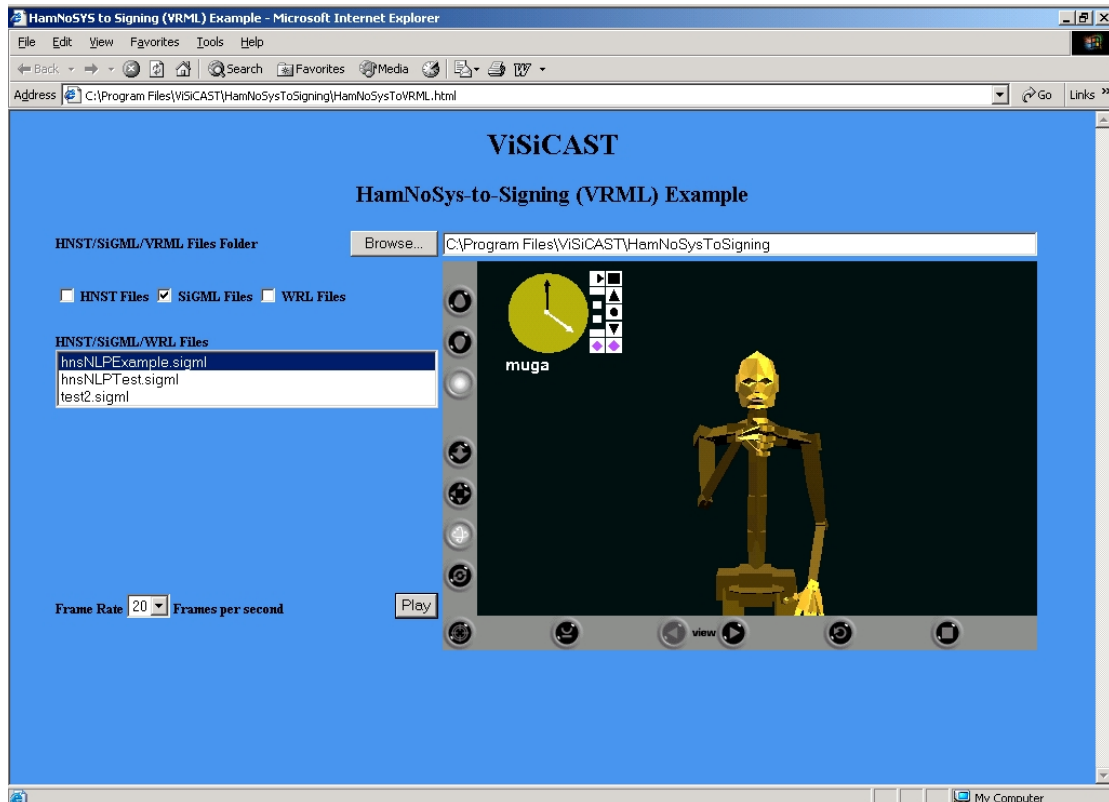


If a HamNoSys token (HNST) file is selected, and the user clicks on 'Play', the application also runs the Java utility which takes an HNST file as input and outputs a SiGML file. The SiGML file is then passed to the SiGMLToBAF control, which creates a BAF file. The BAF player then plays the BAF File. If a SiGML file is selected then this is passed to the SiGMLToBAF control, which creates a BAF File, which is then played by the BAF player. If a BAF file is selected this is simply played by the BAF player.

The list-box allows the selection of one or more files. Each selected file is converted, if required, as appropriate and then played by the BAF player.

VRML Player

A web page was created that allowed the creation of VRML files instead of BAF files. A screen shot of this page follows:



The web page creates the VRML Player control and the SiGMLToVRML control by using the <OBJECT> tag and the appropriate class id. Javascript is used to interact with the controls and to provide the functionality of the user interface.

The list-box only allows a single item to be selected, as the VRML player can only play a single VRML file at a time.

User Instructions

Installation

- 1) Install ViSiCAST Avatar Controls (which includes MSXML3). Note: only the 'Visicast ActiveX Controls' component needs to be installed.
- 2) Install ActivePerl v5.6.1 (ActivePerl-5.6.1.626-MSWin32-x86-multi-thread.msi). Note that the path of the folder '..\Perl\bin' must be added to the PATH environment variable.
- 3) Install Java JDK 1.3.x (j2sdk-1_3_1-win.exe).
- 4) Install Cortona VRML Client v3.1 (cortvrml.exe).
- 5) Restart the PC.
- 6) Run setup.exe.

Running the BAF example

- 1) Select menu option "Start->Programs->HamNoSysToSigning->HamNoSys To BAF", to run the BAF example.
- 2) Click on the 'HNST Files', 'SiGML Files' or 'BAF Files' check-box and select one or more files in the list-box.
- 3) Click on the 'Play' button.
 If the 'HNST Files' check-box is checked then a SiGML file with the same name is created, and then a BAF file with the same name is created, and then played for each selected HNST file.
 If the 'SiGML Files' check-box is checked then a BAF file with the same name is created, and then played for each selected HNST file.
 If the 'BAF Files' check-box is checked then each selected file is played.

So, if 'test.hnst' is selected and the 'Play' button is clicked then a file named 'test.sigml' is created, and then a file named 'test.baf' is created and played.

The status bar informs of the file being created or played at each stage.

Running the VRML example

- 1) Select menu option "Start->Programs->HamNoSysToSigning->HamNoSys To VRML", to run the VRML example.
- 2) Click on the 'HNST Files', 'SiGML Files' or 'VRML Files' check-box and select a file in the list-box.
- 3) Click on the 'Play' button.
If the 'HNST Files' check-box is checked then a SiGML file with the same name is created, and then a VRML file with the same name is created, and then played.
If the 'SiGML Files' check-box is checked then a VRML file with the same name is created, and then played.
If the 'VRML Files' check-box is checked then that file is played.

So, if 'test.hnst' is selected and the 'Play' button is clicked then a file named 'test.sigml' is created, and then a file named 'test.vrml' is created and played.

The status bar informs of the file being created at each stage.

Notes

- a) The HNST files must be in D5-3 format.
- b) The generation of a BAF file may take a while.

Moving the BAF Playing Avatar

Positioning the mouse on the avatar display panel and dragging upwards causes the user's viewpoint to move further away from the Avatar, so the Avatar appears to recede into the distance. Conversely, dragging downwards causes the user's viewpoint to move towards the Avatar, so the Avatar appears to move closer.

A leftward drag causes the user's viewpoint to rotate to the Avatar's left, that is, the Avatar appears to turn to its right. A movement to the right causes the user's viewpoint to rotate to the Avatar's right, so the Avatar appears to turn to its left.

With the 'Ctrl' key pressed, an upward drag of the mouse causes the user's viewpoint to move downwards, that is, the Avatar apparently moves upwards. Similarly, Ctrl-Dragging downwards causes the user's viewpoint to move upwards, so the Avatar appears to move downwards.

Note that the Avatar cannot be moved while a BAF file is being generated.

Synthetic Animation of Deaf Signing Gestures

Richard Kennaway

School of Information Systems, University of East Anglia**

Abstract

We describe a method for synthesizing deaf signing animations from a high-level description of signs in terms of the HamNoSys transcription system.

1 Introduction

1.1 Background

The object of the ViSiCAST project is to facilitate access by deaf citizens to information and services expressed in their preferred medium of sign language. ViSiCAST aims to provide support in three distinct application areas: broadcasting, face-to-face transactions, and the World-Wide Web (WWW). A central feature of the project is its use of computer-generated virtual humans, or avatars, to present deaf signing; hence, the technical activity of the project focuses on two areas: language processing technology and avatar technology. For an introductory account of the whole project the reader is referred to [2].

In outline, the task of signing textual content is decomposed into the following sequence of transformations:

1. from text to semantic representation;
2. from semantic representation to a morphological representation, which latter is sign-language specific;
3. from morphology to a signing gesture notation;
4. from signing gesture notation to avatar animation.

This paper deals with the last of these steps, and describes an initial attempt to create synthetic animations from a gesture notation, to supplement or replace our current use of motion capture.

** School of Information Systems, University of East Anglia, Norwich, NR4 7TJ, U.K.
Email: jrk@sys.uea.ac.uk

* We acknowledge funding from the European Union under the Framework V IST Programme (Grant IST-1999-10500).

1.2 Motion capture vs. synthetic animation

ViSiCAST has developed from two previous projects, one in broadcasting, *Sign-Anim* (also known as *Simon-the-Signer*) [8, 9, 14], and one in Post Office transactions, *Tessa* [7]. Both these applications use a signing avatar system based on *motion capture*: before a text can be signed by the avatar, the appropriate lexicon of signs must be constructed in advance, each sign being represented by a data file recording motion parameters for the body, arms, hands, and face of a real human signer. For a given text, the corresponding sequence of such motion data files can be used to animate the skeleton of the computer-generated avatar. The great merit of this system is its almost uncanny authenticity: even when captured motion data is “played back” through an avatar with physical characteristics very different from those of the original human signer, the original signer (if already known to the audience) is nevertheless immediately recognizable in the result.

On the other hand, motion capture is not without drawbacks.

- There is a substantial amount of work involved in setting up and calibrating the equipment, and in recording the large number of signs required for a complete lexicon.
- It is a non-trivial task to modify captured motions.

There are several ways in which we would wish to modify the raw motion capture data. Data captured from one signer might be played back through an avatar of different body proportions. One might wish to change the point in signing space at which a sign is performed, rather than recording a separate version for every place at which it might be performed. Signs recorded separately, and perhaps by different signers, need to be blended into a continuous animation. Much research exists on algorithmically modifying captured data, though to our knowledge none is concerned specifically with signing, a typical application being modification of a walking character’s gait to conform to an uneven terrain. An example is Witkin and Popović’s “motion warping” [10, 15]. We are therefore also interested in synthetic animation: generation of the required movements of an avatar from a more abstract description of the gestures that it is to perform, together with the geometry of the avatar in use. The animation must be feasible to generate in real time, within the processing power available in the computers or set-top boxes that will display signing, and the transmitted data must fit within the available bandwidth.

Traditional (i.e. non-computer) animation is a highly laborious process, to which computers were first introduced to perform in-betweening, creating all the frames between the hand-drawn keyframes. Even when animations are entirely produced on computer, keyframes are still typically designed by hand, but using 3D modelling software to construct and pose the characters. In recent years, physics-based modelling has come into use, primarily for animating inanimate objects such as stacks of boxes acted on by gravity. Synthetic animation of living organisms poses a further set of problems, as it must deal not only with gravity and the forces exerted by muscles, but also with the biological control systems

that operate the muscles. It is only in the last few years that implementation techniques and fast hardware have begun to make synthetic real-time animation of living movement possible. The tutorial courses [3, 4] give an overview of this history and the current state of the art.

We present here a simplified biomechanical model dealing with the particular application of signing and the constraints of real-time animation.

2 Description of signs

We start from the HamNoSys [12] notation for transcribing signing gestures developed by our partners at IDGS, University of Hamburg. For use in the ViSi-CAST project, we have developed a version of HamNoSys encoded in XML [1], called SiGML (Signing Gesture Markup Language). This is an alternative syntactic representation to facilitate computer processing. As HamNoSys has been designed to be read by humans rather than computers, we use the former for the presentation of examples in this paper.

HamNoSys breaks each sign down into components such as hand position, hand orientation, hand shape, motion, etc. The current version only records the manual components of signs; the recording of facial components is a subject of current research by its developers. When a notation for facial components is available we intend to extend the work reported here to include them. We will not attempt to describe HamNoSys in detail (see the cited manual), but indicate its main features and the issues they raise for synthetic animation.

A typical HamNoSys transcription of a single sign is displayed in Figure 1. This is the DGS (German Sign Language) sign for “GOING-TO”. The colon sign

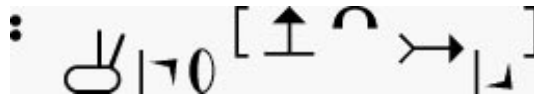


Fig. 1. HamNoSys transcription of the DGS sign for GOING-TO

specifies that the two hands mirror each other. The next three glyphs specify the starting position: the finger and thumb are extended with the other fingers curled, the index finger pointing up and forwards, and the palm of the right hand facing to the left. The part in square brackets indicates the motion: forwards, in an arc curved in the vertical plane, while changing the orientation of the hand so that the index fingers point forwards and down. HamNoSys describes the physical action required to produce the sign, not the sign’s meaning.

Note that HamNoSys describes signs in terms of basic concepts which are not themselves given a precise meaning: “close to”, “chest level”, “fast”, “slow”, etc. Other aspects are not recorded at all, and are assumed to take on some “default” value. For example, HamNoSys records the positions of the hands, but not of the rest of the arms. Shoulder shrugs or raising of the elbows can be notated, but

for signs which do not require such movements, the positions of shoulders and elbows are omitted, and are assumed to be in an ordinarily relaxed position. This is deliberate: only those parts of the action are transcribed which are required to correctly form the sign, and only with enough precision as is necessary. People learning to sign learn from example which parts of the action are significant, and how much precision is required for good signing. To synthesise an animation from a HamNoSys description requires these choices to be made by the animation algorithms.

3 Synthesis of static gestural elements

3.1 Disambiguation

We illustrate by an example how we have approached the task of making precise the fuzzy definitions of HamNoSys components.

HamNoSys defines a set of 60 positions in the space in front of the signer. These are arranged in a grid of four levels from top to bottom, five from left to right, and three from near to far. The four vertical levels are indicated by the glyphs $\overline{\square}$ (shoulder level), $\overline{\square}$ (chest level), $\overline{\square}$ (abdomen level), and $\overline{\square}$ (below abdomen level). For any given avatar, we define these to be respectively at heights s , $(s + e)/2$, e , and $(e + w)/2$, where s , e , and w are the heights of the shoulder, elbow, and wrist joints of the standing avatar when both arms hang vertically. We have defined these heights in terms of the arms rather than the torso, because these measurements are guaranteed to be present for any avatar used for signing.

HamNoSys indicates left-to-right location by a modification to these glyphs: the five locations at chest level are represented by the glyphs: $\overline{\square}$, $\overline{\square}$, $\overline{\square}$, $\overline{\square}$, and $\overline{\square}$. We define their left-to-right coordinates in terms of the positions of the shoulders: centre is midway between the shoulders, and the points left and right are regularly spaced with gaps of 0.4 times the distance between the shoulders.

The three distances from the avatar are notated in HamNoSys by \rangle (near), no explicit notation for neutral, and \curvearrowright (far); we defined these in terms of the shoulder coordinates and the length of the forearm.

An important feature of this method of determining numerical values is that it is done in terms of measurements of the avatar's body, and can be applied automatically to any humanoid avatar. The precise choice of coordinates for these and other points must be judged by the quality of the signing that results. The animation system is currently still under development, and we have not yet brought it to the point of testing.

Hand orientations are described by HamNoSys in terms which are already precise: the possibilities are all of the 26 non-zero vectors (a, b, c) , where each of a , b , and c is -1 , 0 , or 1 . When more precision is required, HamNoSys also allows the representation of any direction midway between two such vectors. These directions are the directions the fingers point (or would point if the fingers were extended straight); the orientation of the palm around this axis takes 8 possible values at 45 degree intervals.

To complete the specification of HamNoSys positions requires definitions along similar lines of all the positions that are significant to HamNoSys. In addition to these points in space, HamNoSys defines many contact points on the body, such as positions at, above, below, left, or right of each facial element (eyes, nose, cheeks, etc.), and several positions on each finger and along the arms and torso. The total number of positions nameable in HamNoSys comes to some hundreds. Any avatar for use in signing must include, as part of its definition, not only the geometry of its skeleton and surface, and its visual appearance, but also the locations of all of these “significant sites”.

3.2 Inverse Kinematics

Given a definition of the numerical coordinates of all the hand positions described by HamNoSys, we must determine angles of the arm joints which will place the hand in the desired position and orientation. This is a problem in “inverse kinematics” (forward kinematics being the opposite and easier problem, of computing hand position and orientation from the arm joint angles).

The problem can mostly be solved by direct application of trigonometric equations. Two factors complicate the computation: firstly, the arm joint angles are not fully determined by the hand, and secondly, care must be taken to ensure that physiologically impossible solutions are avoided.

If the shoulder joint remains fixed in space, and the hand position and orientation are known, then one degree of freedom remains undetermined: the arm can be rotated about the line from the shoulder to the wrist joint. If the sign being performed requires the elbow to be unusually elevated, this will be notated in the HamNoSys description; otherwise, a choice must be made by the animator as to what is a natural position for the elbow. The first solution we adopted required the elbow to lie vertically below the line from shoulder to wrist. This gives satisfactory results for placements of the hand in the same half of signing space as the shoulder, but reaches across the body resulted in the upper arm penetrating the torso. A correction was therefore made to rotate the elbow away from the body when necessary to maintain a certain minimum separation. In addition, for such reaches, and for reaches into the “far” part of signing space, greater realism is obtained by using the sternoclavicular joint to let the shoulder move some distance towards the target point.

For positions around the head, further care must be taken to avoid the hand penetrating the head. It should be noted that HamNoSys itself does not attempt to syntactically exclude the description of physiologically impossible signs. One can, for example, specify a hand position midway between the ears. This is not a problem; the real signs that we must animate are by definition possible to perform. This implies that a synthetic animation system for signing does not have to solve general collision problems (which are computationally expensive), but only a few special cases, such as the elbow positioning described above.

3.3 Contacts

Besides specifying a hand position as a point in space or on the body, HamNoSys can also notate contacts between parts of both hands. The BSL two-handed spelling signs are an example of these. The inverse kinematic problem of calculating the arm angles necessary to bring the two hand-parts into contact can be reduced to two one-arm problems, by determining for each arm separately the joint angles required to bring the specified part of the hand to the location in space at which the contact is required to happen.

This is an area in which motion capture has difficulty, due to the accuracy with which some contacts must be made. A contact of two fingertips, for example, may appear on playback of the raw data to pass the fingers through each other, or to miss the contact altogether. This is due to basic limitations on the accuracy of capture equipment. When using motion capture data we deal with this problem by editing the calibration data for the equipment in order to generate the correct motion.

3.4 Handshapes

HamNoSys distinguishes around 200 handshapes. At present, we are implementing these simply by specifying the angles of all the joints of the hand for each handshape, a tedious but routine task. There is a certain amount of structure to the class of handshapes which reduces the effort: for example, a given bend of the index finger may occur in many different handshapes. For the best quality of hand shape, the joint angles should be generated algorithmically from a knowledge of the geometry of the particular avatar's hands.

4 Motion synthesis

We have so far discussed how to synthesise a static gesture. Many signs include motion as a semantic component, and even when a sign does not, the signer must still move to that sign from the preceding sign, and then move to the next.

If we calculate the joint angles required for each static gesture, and then linearly interpolate over time, the effect is robotic and unnatural. Artificial trajectories can be synthesised (e.g. sine curve, polynomial, Bezier, etc.), but we take a more biologically based approach and model each joint as a control system.

For the particular application of signing, the modelling problem is somewhat easier than for general body motion, in that accurate physics-based modelling is largely unnecessary. Physics plays a major role in motions of the lower body, which are primarily concerned with balancing and locomotion against gravity. This is also true of those motions of the upper body which involve exertions such as grasping or pushing. Signing only requires movement of upper body parts in space, without interaction with external objects or forces. The effect of gravity in shaping the motion is negligible, as the muscles automatically compensate.

We therefore adopt a simplified model for each joint. The distal side of the joint is represented as a virtual mass or moment of inertia. The muscles are

assumed to exert a force or torque that depends on the difference between the current joint angle and the joint angle required to perform the current sign, the computation being described in detail below.

Simplifications such as these are essential if the avatar is to be animated in real time.

4.1 A brief introduction to control systems

This brief summary of control theory is based on [11]. We do not require any mathematics.

A *control system* is any arrangement designed to maintain some variable at or near a desired value, independently of other forces that may be acting on it.

In general, a control system consists of the following parts:

1. The *controlled variable*, the property which the controller is intended to control.
2. A *perception* of the current value of the controlled variable.
3. A *reference signal* specifying the desired value of the controlled variable.
4. An *error signal*, being the difference between the reference and the perception.
5. The *output function*, which computes from the error signal (and possibly its derivatives or integrals) the *output signal*, which has some physical effect.

The effect of the output signal in a functioning control system is to bring the value of the controlled variable closer to the reference value. The designer of a real (i.e. non-virtual) control system must choose an output function which will have this effect. For the present application, our task is to choose an output function such that, when the reference angle for a joint is set to a new value, the joint angle changes to reach that value in a realistic manner.

4.2 Hinge joints

Our application of this avatar animation places a controller in each joint. For a hinge joint such as the elbow, the controlled variable is the angle of the joint. The reference value is the angle which is required in order to produce some gesture. The perception is the current angle. The output is a virtual force acting on the distal side of the joint. The latter is modelled as a mass, whose acceleration is proportional to the force. We also assume a certain amount of damping, that is, a force on the mass proportional to and in the opposite direction to its velocity.

The mass, the force, and the damping are fictitious, and not intended as an accurate physical model; their values are tuned to provide a realistic-looking response to changes in the reference value.

Figure 2 illustrates the response of this system to a series of sudden changes in the reference value. For a sequence of static gestures, the reference value will change in this manner, taking on a new value at the time when the next gesture is to be performed.

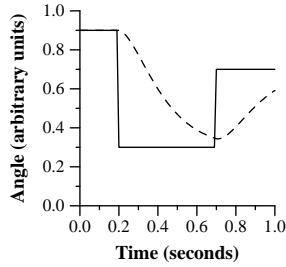


Fig. 2. Solid line: reference value. Dashed line: current value.

4.3 Higher degree joints

A turret or universal joint has two degrees of freedom. An example is the joint at the base of each finger, which can move the finger up and down, or left and right, but cannot rotate the finger about its own axis. This can be modelled as a pair of hinge joints at right angles, and the method of the preceding section applied to each hinge. This will not be accurate if the rotation of either hinge approaches a right angle, but the universal joints in the upper body all have sufficiently limited mobility that this is not a problem.

A ball and socket joint such as the shoulder has three degrees of freedom. In principle, it can be modelled by three hinge joints in series. However, there is no obvious way to choose axes for the hinges that corresponds to the actual articulation of the muscles. Mathematically, there are singularities in the representation, which correspond to the physical phenomenon of “gimbal lock”, which does not occur in a real shoulder joint. Aesthetically, animation of the three hinge angles tends to give wild and unnatural movements of the arm.

Instead, we reduce it to a single one-dimensional system. If the current rotation of the shoulder joint is q , and the required rotation is q' , we determine a one-dimensional trajectory between these two points in the space of three-dimensional rotations. The trajectory is calibrated by real numbers from 0 to 1, and this one-dimensional calibration used as the controlled variable. When the reference value is next changed, a new trajectory is computed and calibrated.

4.4 Moving signs

Some moving signs are represented in HamNoSys as a succession of static postures. They may also be described as motion in a particular direction by a particular amount, with or without specifying a target location. These can be animated by considering them as successive static postures, in the latter case calculating a target location from the direction and size of the movement.

HamNoSys can also specify the tempo of the motion or its path, when the path is not the default straight line motion. If the tempo is unmarked, it is performed in the “default” manner, which is what we have attempted to capture by our control model of motion. It may also be fast or slow (relative to the general

tempo of the signer), or modulated in various ways such as “sudden stop” or “tense” (as if performed with great effort). These concepts are easily understood from example by people learning to sign. Expressing them in terms of synthesised trajectories is a subject of our current work.

4.5 Sequences of signs

Given a sequence of signs described in HamNoSys or SiGML, and the times at which they are to be performed, we can generate a continuous signing animation by determining the joint angles required by each sign, and setting the reference values for the joint controllers accordingly at the corresponding times (or slightly in advance of those times to account for the fixed lag introduced by the controllers). The blending of motion from each sign to the next is performed automatically, without requiring the avatar to go to the neutral position between signs.

4.6 Ambient motion

Our current avatar has the ability to blend signing animation data with “ambient” motion — small, random movements, mainly of the torso, head, and eyes — in order to make it appear more natural. This can be used even when the animation data come from motion capture. It is particularly important for synthetic animation, since we only synthesise movements of those joints which play a part in creating the sign. If the rest of the body does not move at all — and there are few signs which require any torso motion as part of their definition — the result will look unnaturally stiff. Motion capture data can be blended with synthesized data; a possible future approach is to generate these small random movements algorithmically.

5 Target platform

These ideas were initially prototyped in VRML 97, the Virtual Reality Modelling Language [5], with a ball-and-stick avatar constructed from the H-anim specification for virtual humanoids [13] (Figure 3(a)). The control algorithms were embedded into the VRML model. Animations are thus computed at run time, not precalculated. As described above, HamNoSys handshapes were converted to joint angles by hand, and hand positions in the signing space in front of the avatar converted to arm joint angles by inverse kinematic calculations.

A ball-and-stick avatar is not suitable for production-quality signing, but it gives the animator a much clearer view of the motion. As the avatar is H-anim compliant, we were able to cut and paste several other H-anim avatars available on the Web, recompute the inverse kinematics for the new avatar’s dimensions, and generate more realistic-looking animations, at the cost of reduced frame rate (Figure 3(b)).

This was useful as a prototyping exercise. However, current VRML viewers and available hardware are not fast enough to provide the frame rate of 15 to 25 frames/second required for readable signing. (25 fps is the frame rate of European broadcast television; the lower bound of 15 fps was communicated to us by Thomas Hanke.)

The avatar currently in use by ViSiCAST is called Visia, and was developed by Televirtual Ltd., one of the ViSiCAST partners. It is not H-anim compliant, but adopts a similar internal structure of a hierarchical skeleton of bones. It can be driven by a stream of information about bone rotations, positions, and lengths (although it is primarily the rotations that change from one frame to another). The avatar automatically adjusts its seamless skin to fit the bones. Visia is illustrated in Figure 3(c).

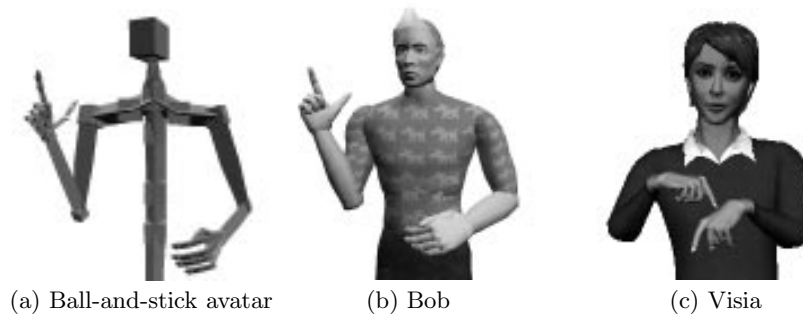


Fig. 3. H-anim avatars

There is a fairly simple correspondence between H-anim joints in the torso, arms, and hands, and Visia's bones, which allows us to generate motion data for both targets with little change to the code. Visia includes many more face bones than H-anim, but synthetic facial animation is the subject of future work. A third possible target is BAP data (Body Animation Parameters), a part of the MPEG-4 standard concerned with animation of humanoid figures [6], and closely connected with H-anim.

6 Conclusions

We have described above the design and initial implementation of a method of automatic synthesis of manual signing gestures from their transcriptions in the HamNoSys/SiGML notations. We are confident that the approach described here will produce results that compare favourably with existing alternatives. Perhaps the most interesting comparison will be with the system based on motion capture, as already used in ViSiCAST. As our synthetic signing can target the same avatar model as the motion capture system, this provides the opportunity to undertake two kinds of comparison. Firstly, we will be able to do a meaningful

comparison of user reaction to our synthetic signing with reaction to signing based on motion capture. In addition, as our synthesis process drives the avatar via a stream of data whose form is identical to that produced from motion-capture, we are also in a position to perform quantitative comparisons between the two methods. In particular, if the evidence warrants it, we could consider a hybrid approach, combining synthetic generation with elements of motion-captured data.

7 Acknowledgements

We are grateful for funding from the EU, under the Framework V IST Programme (Grant IST-1999-10500). We are also grateful to colleagues, both here at UEA, and at partner institutions, for their support.

The “Bob” avatar of Figure 3(b) is available at <http://ligwww.epfl.ch/~babski/StandardBody/>, and is due to Christian Babski and Daniel Thalmann at the Computer Graphics Lab at the Swiss Federal Institute of Technology. Body design by Mireille Clavien.

References

1. D. Connolly. *Extensible Markup Language (XML)*. World Wide Web Consortium, 2000.
2. R. Elliott, J.R.W. Glauert, J.R. Kennaway, and I. Marshall. The development of language processing support for the ViSiCAST project. In *ASSETS 2000 - Proc. 4th International ACM Conference on Assistive Technologies, November 2000, Arlington, Virginia*, pages 101–108, 2000.
3. Andrew Glassner. Introduction to animation. In *SIGGRAPH '2000 Course Notes*. Assoc. Comp. Mach., 2000.
4. Jessica Hodgins and Zoran Popović. Animating humans by combining simulation and motion capture. In *SIGGRAPH '2000 Course Notes*. Assoc. Comp. Mach., 2000.
5. The VRML Consortium Incorporated. *The Virtual Reality Modeling Language: International Standard ISO/IEC 14772-1:1997*. 1997. <http://www.web3d.org/Specifications/VRML97/>.
6. R. Koenen. *Overview of the MPEG-4 Standard*. ISO/IEC JTC1/SC29/WG11 N2725, 1999. <http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm>.
7. M. Lincoln, S.J. Cox, and M. Nakisa. The development and evaluation of a speech to sign translation system to assist transactions. In *Int. Journal of Human-computer Studies*, 2001. In Preparation.
8. I. Marshall, F. Pezeshkpour, J.A. Bangham, M. Wells, and R. Hughes. On the real time elision of text. In *RIFRA 98 - Proc. Int. Workshop on Extraction, Filtering and Automatic Summarization, Tunisia*. CNRS, November 1998.
9. F. Pezeshkpour, I. Marshall, R. Elliott, and J. A. Bangham. Development of a legible deaf-signing virtual human. In *Proc. IEEE Conf. Multi-Media, Florence*, volume 1, pages pp333–338, 1999.
10. Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proc. SIGGRAPH '99*, pages 11–20. Assoc. Comp. Mach., 1999.

11. W. T. Powers. *Behavior: The Control of Perception*. Aldine de Gruyter, 1973.
12. S. Prillwitz, R. Leven, H. Zienert, T. Hanke, J. Henning, et al. *HamNoSys Version 2.0: Hamburg Notation System for Sign Languages — An Introductory Guide*. International Studies on Sign Language and the Communication of the Deaf, Volume 5. University of Hamburg, 1989. Version 3.0 is documented on the Web at <http://www.sign-lang.uni-hamburg.de/Projects/HamNoSys.html>.
13. B. Roehl. *Specification for a Standard VRML Humanoid*. H-ANIM WG, U.Waterloo, Canada, 1998. <http://ece.uwaterloo.ca/~hh-anim/spec.html>.
14. M. Wells, F. Pezeshkpour, I. Marshall, M. Tutt, and J. A. Bangham. Simon: an innovative approach to signing on television. In *Proc. Int. Broadcasting Convention*, 1999.
15. Andrew Witkin and Zoran Popović. Motion warping. In *Proc. SIGGRAPH '95*, 1995.