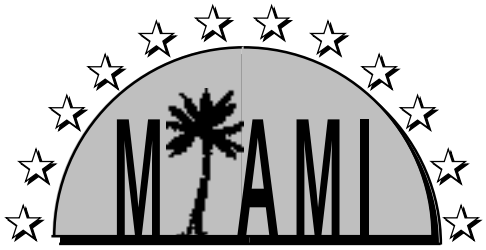


DI 3 - Development of a System Architecture for the Acquisition, Integration, and Representation of Multimodal Information

A Report of the Esprit Project 8579 MIAMI – WP 3 –

March, 1996

Written by

K. Hartung[5], S. Münch[6], L. Schomaker[3],

T. Guiard-Marigny[2], B. Le Goff[2],

R. MacLaverty[4], J. Nijtmans[3], A. Camurri[1] I. Defée[4], C. Benoît[2]

([1]DIST, [2]ICP, [3]NICI, [4]RIIT, [5]RUB, [6]UKA)

Introduction

Workpart 3 follows the experiments on humans in Workparts 1 and 2 and aims at defining a generalized architecture for multimodal data acquisition and representation. This will lead to a common integrated software library which will be employed in the two demonstrator scenarios in Workpart 4. In chapter 1 the common software infrastructure for development is described. Partly, we could make use of existing systems (e.g., X11), but by necessity, also a number of specific MIAMI solutions had to be developed. As already partly covered in our Report on Workpart 2 [15], the basic software architecture in MIAMI revolves around an (1) event-driven setup, handled by the script language and toolkit Tcl/Tk, which also handles (2) the case of 2-D widgets in the user interface, (3) OpenGL for 3-D graphics, and (4) PVM for inter-process control.

At the next, higher, software level, described in chapter 2 and chapter 3, a modular architecture is introduced. This modular concept, which has been developed through intense cooperation between partners in MIAMI has thoroughly redefined the implementation aspects of the originally envisaged work tasks. The goal of the setup is to make available in the consortium a number of 'Lego'-brick components with an integrated multimodal functionality which provides much more than, e.g., a module at the device driver level, but which are at the same time generic enough to fit in a number of different application main programs. As such, these modules will prove to be useful in both the Analogical and the Symbolical demonstrator.

A basic building block, for example, is the realised *Meta-Device Driver (*MDD*)*. This driver allows to connect a wide number of human-movement transducers, such as the mouse, the pen, force joysticks and other relatively low-bandwidth transducers to an application. Another example is a 2-D gesture recognizer (*GESTE*) which may receive planar position coordinates from *MDD*, in order to classify isolated movement patterns and map them to a single symbol or action. A third typical building block is *ICP-FACE-ANIM*, which allows for the visualisation of a talking face, based on flat-text input. Furthermore, this report contains chapters on navigation (chapter 5) and on the ultimate demonstrators (chapter 7).

Contents

Chapter 1

Development Tools for Multimodal Systems

At the lowest software infrastructure level, a number of choices had to be made, and specific MIAMI solutions had to be developed. The goal of these activities, broadly encompassing Worktasks 3.1, but also WT. 4.1.1 and WT. 4.2.1, was to ensure a compatible platform amongst the partners for experimentation with multimodality.

1.1 Basic Tools

Here, the basic tools which are used for most software developments within MIAMI will be described very briefly. We start with the introduction of the basic window system (1.1.1) before describing Tcl/Tk (1.1.2), a toolkit for programming 2D graphics and user interfaces. For realizing 3D graphics, more specialized libraries which have become a kind of standard within the last years have been selected (1.1.3 and 1.1.4). Finally, in order to realize distributed systems and to provide a basic communication mechanism, we decided to use the socket-based PVM protocol (1.1.5).

1.1.1 X—the basic window system

The *X Window System*, also called X or X11, is a network-oriented, hardware-independent window system for workstations. Its main feature is its distributed client-server architecture which is based on a protocol between the workstation and the application.

The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the

most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well. X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays.

In the *X Window System Protocol* (or X protocol), four different data formats are defined: *Requests* are sent from the client to the server, which will return a *reply*, if necessary. In addition, the server sends *events* to the client, usually representing user interactions. Finally, the server signals *errors* if they occur.

The interface between the X protocol and any application using X is provided by the *Xlib*. In this C library, all functions needed to create and manipulate windows, process events, etc. are defined. X itself does not provide a widget set (see 1.1.2), but only supports *basic* mechanisms to display windows.

Windows are the only visual output medium for the clients. In principle, a window is a rectangular region of the screen, used to display text or graphics. Windows may overlap, include another, etc. They are managed by the server and are stored in a tree-like structure.

From the client's point of view, the *events* generated by the server are the primary source of input. All user interactions are modeled as events: keystrokes, mouse movements, button presses, etc. Overall, there exist 33 events, but the following are especially interesting with respect to the detection of action patterns: MotionNotify (cursor movement), EnterNotify (cursor has entered a window), LeaveNotify (cursor has left a window), ButtonPress (mouse button has been pressed), and ButtonRelease (mouse button has been released).

1.1.2 Tcl/Tk—the GUI toolkit

Tcl and Tk are two software packages which provide a programming system for developing and using graphical user interface (GUI) applications [20]. Tcl (tool command language) is a simple scripting language for controlling and extending applications. It provides generic programming facilities and is embeddable. The Tcl interpreter is a library of C procedures which can easily be extended and included in application programs.

The most popular extension of Tcl is Tk, a toolkit for the X Window System. It provides a widget set for building user interfaces written in Tcl, and is also a library of C procedures with the same advantages as Tcl. Tk's widget set is directly based on the Xlib. Both languages can be easily used in combination with other programming languages like C or Lisp, and they are available on almost any hardware platform and operating system which supports X.

Tk provides 15 different classes of widgets like toplevels, frames, buttons, menus, etc. In this context, the *event binding* mechanism is of special interest[1]. Besides the default bindings provided by Tk, the application can install event bindings which can be any kind of Tcl script (i.e., any kind of *program*!). Even more important, this can be done in 'foreign' Tcl/Tk applications as well, because Tk provides the command 'send' for "invoking arbitrary Tcl scripts in any other Tk application on the display; these commands can both retrieve information and also take actions that modify the state of the target application."

1.1.3 OpenGL—low-level 3D graphics

The OpenGL graphics system is a software interface to graphics hardware designed by Silicon Graphic Inc., to allow interactive programs to be created to produce color images of moving three-dimensional objects [18].

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; so a special library, depending on the windowing system of the hardware used, has to be added to OpenGL package. Today, OpenGL is available on SGI under IRIX 5.3 , SUN under Solaris, PC under Win95 or Windows NT.

OpenGL is designed to work efficiently even if the computer that displays the graphics created isn't the computer that runs the graphics program. This might be the case in a networked computer environment where many computers are connected to one another by wires capable of carrying digital data. In this situation, the computer on which the program runs and issues OpenGL drawing commands is called the client, and the computer that receives those commands and performs the drawing is called the server. The format for transmitting OpenGL commands (called the protocol) from the client to the server is always the same, so OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server (compare to 1.1.1).

OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. With OpenGL, a desired model must be built up from a small set of geometric primitive: points, lines, and polygons. A sophisticated library that provides these features could certainly be built on top of OpenGL, in fact, that's what OpenInventor is.

[1]The events known to Tcl/Tk are very similar to those supported by the Xlib, see 1.1.1.

The major graphics operations performed by OpenGL and necessary to render an image on the screen are the following:

1. Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.)

2. Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene.

3. Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, or obtained by pasting a texture onto the objects.

4. Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called rasterization.

All these stages will be detailed later in chapter 4 (Representation of Object Space) after a general introduction to Computer Graphics.

1.1.4 OpenInventor—a 3D graphics library

OpenInventor is an object-oriented toolkit based on OpenGL that provides objects and methods for creating interactive three-dimensional graphics applications. Available from Silicon Graphics and written in C++, OpenInventor provides pre-built objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats [29, 30, 31].

OpenInventor is a set of building blocks that allows to write programs that take advantage of powerful graphics hardware features with minimal programming effort. The toolkit provides a library of objects that can be used, modified, and extended to meet new needs. Inventor objects include database primitives, including shape, property, group, and engine objects; interactive manipulators, such as the handle box and trackball; and components, such as the material editor, directional light editor, and examiner viewer.

The Inventor toolkit is—like OpenGL—independent of the underlying window system. So a component library is helpful for using Inventor with specific window systems.

Inventor focuses on creating 3D objects. All information about these objects (their shape, size, coloring, surface texture, location in 3D space) is stored in a scene database. This

information can be used in a variety of ways. The most common one is to display, or render, an image of the 3D objects on the screen.

Because the Inventor database holds information about the objects as they exist in their own 3D "world," not just as a 2D array of pixels drawn on the screen, other operations in addition to rendering can be performed on the objects. The objects in the scene can be picked, highlighted, and manipulated as discrete entities. Bounding-box calculations can be performed on them. They can be printed, searched for, read from a file, and written to a file. Each of these built-in operations opens up a flexible and powerful arena for the application programmer. In addition, this programming model is intuitive because it is based on the physical and mechanical world we live in.

The node is the basic building block used to create three-dimensional scene databases in Inventor. Each node holds a piece of information, such as a surface material, shape description, geometric transformation, light, or camera. All 3D shapes, attributes, cameras, and light sources present in a scene are represented as nodes.

After having constructed a scene graph, a number of operations or actions can be applied to it, including rendering, picking, searching, computing a bounding box, and writing to a file. A scene graph can also be read from a file which is very useful for using the same 3D scene in different programs

A detailed description of how to build a 3D scene with OpenInventor will be given in chapter 4 (Representation of Object Space) after a general introduction to Computer Graphics.

1.1.5 PVM—exchanging data via sockets

PVM (parallel virtual machine) is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. Although primarily designed to support easy and flexible interprocess communication via sockets in distributed environments, PVM is a software package which supports the design of applications realized as independent modules running as single processes. In this case, PVM provides all procedures needed to exchange any kind of data between these processes, distributed or not [8]. Therefore, it is also the ideal tool for implementing *multimodal* and *multi-agent systems*.

Various processes might be organized 'stand-alone' or in groups in the virtual machine. Upon startup, each process is given a unique task identifier (TID) which is used in all following communications to address the process. Data can be sent in various formats (ranging from single bytes to complex strings) either to a single process, to all processes

in a specific group, or—via broadcast—to all processes in the virtual machine.

1.2 Tools Developed within MIAMI

The basic tools described in the previous section provide a very good basis for software development. However, they do not fulfill all the requirements of a multimodal system and do not support all of the functionality needed. Therefore, we have developed some general software packages (which are not meant to be exclusively used in MIAMI) instead of implementing specialized solutions by each partner.

1.2.1 TkPVM—combining Tcl/Tk and PVM

In order to link different modules which may be written in different languages (Tcl/Tk/C) and using different libraries (PVM, MDD, OpenGL) it was necessary to start with a new approach. Each of the modules has its own special functionality which must be used by other modules:

- **Tcl** - Simple scripting language for applications
- **Tk** - 2D graphics library
- **PVM** - Socket library
- *MDD* - I/O library
- *AAS* - audio library
- **OpenGL** - 3D graphics library
- ...(other components, see the next chapters)

A solution to this diversity problem is the use of parallel processes and a communication scheme. We have chosen to use PVM for this concept. If PVM is going to be the glue which links everything together, all modules in MIAMI must have a PVM interface. For C-based modules, the PVM functions can be called directly. But because Tcl and Tk are interpreted, functions must be written to allow two-way communication. Therefore, the following conclusions have been drawn. (a) Applications written in Tcl/Tk must be able to perform PVM-functions. Examples of such functions are:

- spawning of other processes

- writing to other processes
- getting information about other processes

(b) If another process sends information to a Tcl/Tk application, this application should be triggered to do some action. This required some modifications to the Tcl event-loop. In discussion and cooperation with John Ousterhout (the writer of Tcl/Tk) the latest release (Tcl7.5b3) now contains these modifications. So, the latest version of Tkpvm (1.0b3) doesn't require a special MIAMI version of the Tcl interpreter any more. In fact, MIAMI-originating ideas are embedded in the official release.

The most important functions now available in Tkpvm are:

bind prepare the event-loop in such a way that the next time data is received, an action is performed. This is the most important feature, because it allows multiple input sources to be handled 'simultaneously'.

tasks ask information about other tasks

send send data to other tasks

recv receive data from other tasks

spawn start a new task

kill kill a task

All these functions have an equivalent in C. For example the *bind* command can be used from C with the following two functions:

`Tkpvm_CreateEventHandler()`	`register an event-handler to the Tcl event-loop that is called as soon as a specified PVM package arrives`
`Tkpvm_DeleteEventHandler()`	`remove the event-handler from the Tcl event-loop.`

A WWW home page for Tkpvm has been set up, which always contains the latest information: `http://www.nici.kun.nl/tkpvm/`

The Tkpvm package is made available to the public domain in the Internet Parallel Computing Archive which is located in the U.K. and has mirror-sites in Australia, France and Japan. This is a direct product of the MIAMI project, which is already in active use in many institutes (100 Tkpvm ftp file accesses per day on the NICI ftp server alone).

1.2.2 Other extensions to Tcl/Tk

Because Tcl/Tk was originally not designed to work with other input devices, some more MIAMI-based modifications were needed. These can be categorized into installation features and Tcl internals.

Installation of Tcl/Tk

If multiple processes are executing on the same machine, each of them requires a certain amount of memory. If the number of processes increases, this may slow down the machine considerably. Therefore a lot of nowaday machines support the use of shared libraries. Tcl/Tk originally didn't support it, so we made the necessary adaptations for that. The latest release (Tcl7.5b3/Tk4.1b3, which is released March 9 1996) includes this support, which is mainly adapted from the work done for MIAMI.

Tcl event loop

For Tkpvm, a Tcl event loop extension was needed which was not originally supported. The event-loop should handle PVM events in the same way as it would handle X-events. The latest releases of Tcl/Tk include enough support for this, so the adaptations that were made originally are now no longer needed.

Speedup and additional features

When Tk is used to draw on the screen, it is very slow in some situations. The reason is that after every mouse-move, Tk keeps track of which items are covered by the mouse pointer. Therefore all items are given an extra option "-state", which has 3 possible values:

- normal
- disabled
- hidden

In the disabled state screen objects are displayed but cannot be clicked on. This makes the screen update much faster, specially if the display contains many lines with many coordinates.

For synchronization between the Screen and Audio (for example) the delay used in Tk is disastrous. Normally the screen is only updated when there is some idle time, but there is

no way telling when that will happen. Therefore the "-updatecommand" option is added which reports back when the actual screen update takes place.

Tk doesn't supply any form of dashed lines. The only way to do that is to let each segment be a separate line. But that costs very much computing power, while the X11-server already has support for that directly. Therefore the "-dash" option is added to all screen objects.

The only way to change lines is to replace all coordinates with a complete new set of coordinates. But for creating pen drawings, often you only have to add a few extra points to the end of the line. Therefore additional commands have been added to do that. This also has the effect that not the complete line has to be updated on the screen, just the part that changed.

A full description of these additional features can be found at:

`http://www.nici.kun.nl/~nijtmans/tcl/patch.html`

Since the first version of these additions was made public, some other people have contributed more code to it. This patch has become rather popular now, so it is likely that it will become part of the standard Tk distribution one day.

Chapter 2

Signals & Sampling of Natural Modalities

This chapter deals with characteristics of the signals and sampling process of natural modalities and is broadly based on the plans of Worktask 3.1. As described in Chapter 3, we can make a distinction between:

- Input Signals,
- Parameters In,
- Parameters Out,
- Output Signals.

However, a distinction can be made between two types of input streams to a module in MIAMI: (a) an input in the form of a channel carrying a signal from a 'natural modality' (audio, movement, force, video) as captured from the physical world by a **transducer**, and (b) input in the form of (coded) signals or parameters, coming from file or from other modules.

In this chapter we will direct our attention at the characteristics of the signals and the typical sampling processes of the natural modalities *for modules receiving their input from transducers*. We will start with the modality of handwriting and pen movement input as recorded by a 2D transducer such as a tablet or digitizer. This modality is also used to introduce a number of concepts on sampling in applications in general. The same order of sampling aspects will be assessed for a number of other input modalities apart from pen movement.

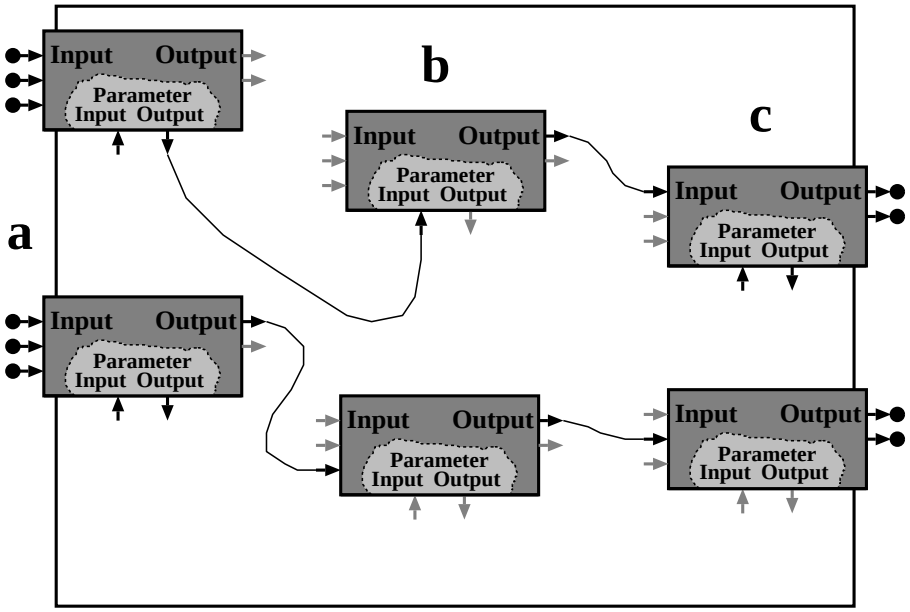


Figure 2.1: A modular multimodal architecture. Modules have Input and Output Signals, and Input/Output Parameters. Examples of Signal In, Parameter Out are the feature extractors for speech or handwriting. An example of Parameter In, Signal Out is the parametrized face. Furthermore we can make a distinction between modules connected to input transducers (a), modules connected to other modules only (b), and modules connected to output effectors (c).

2.1 Handwriting and Pen Movement

2.1.1 Transducer characteristics & sampling requirements

Regardless of the technology of pen-tip position sensing used, a number of requirements for position sensing in pen computing can be defined. The current quality of digitizers is sufficient for different handwriting applications, such as drawing, writing, signature verification and normal point and click actions. Problems of noise and of positional error due to pen-tilt are present but reduced to acceptable levels. It is at the output side (graphical inking on CRT and LCD) where current technology has its limitations. Table 2.1 shows the typical characteristics of pen-tip position sensor devices.

Parameter	*Min.*	*Max.*	*Typical*	*Units*
Sampling rate	50	200 (good)	100	Hz
Resolution	0.02 (good)	0.1	0.02	mm
Accuracy	—	—	0.1	mm

Table 2.1: Characteristics of pen-tip position sensor devices

Figure 2.2 shows a typical power spectral density function for pen-tip movements in handwriting, averaged over 32 writers, 210 words per writer. The word XY pen-tip displacement time functions were circularized with a cosine transition function and padded with zeros until 512 samples (i.e., about 5s of writing time), and an FFT was calculated for that word. The average PSDF over words was calculated by accumulating $|FFT|^2$. The spectrum shows that from the Nyquist sampling theorem point of view, a sampling frequency of 20 samples/second would be sufficient for reconstruction of the signal. However, it is much cheaper to use higher sampling rates than to reconstruct the trajectory and display it in real time. A sampling frequency of 100 samples/second yields about 10 points per stroke in normal handwriting. Five points per stroke is the (barely) acceptable minimum, for users of pen computers.

There are different types of sampling requirements, each with a typical demand on the system resources. There may be input modalities which require continuous sampling for a prolonged period, or modalities in which the sampling occurs in bursts. Given the current state of technology, which is based on graphical user interfaces (GUIs) which handle events at moderate rates ($<$ 100 events/s), it is clear that for high-bandwidth modalities, the ratio of *(Number of samples)/(Number of events)* will be much larger than one.

In pen interfaces, we can make a distinction between a number of different input types, each with a typical time span, as summarized in table 2.2 below:

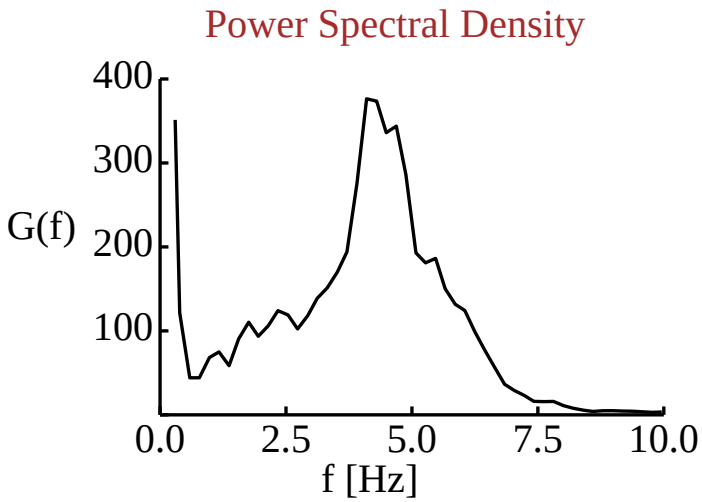


Figure 2.2: The typical power spectral density of pen-tip movement in handwriting

Sampling Burst Duration [s]	*Interaction category*
< 1	gestures and handprint characters
< 10	(cursive) words, signatures
> 10	phrases, drag/drop interaction
Continuous	pen-up movement controls a cursor

Table 2.2: Different sampling-burst durations in the use of the pen as an input device under different application conditions

2.1.2 Sampling priority

Explanation: There is a difference between high-fidelity modalities and modalities with more lenient requirements. For some modalities, the loss of a single sample is fatal, e.g. because it is a nuisance for the user (as in audio) or because it deteriorates pattern recognition functions (as in speech or handwriting recognition). In other modalities, it is often assumed sufficient that the system follows the input approximately, such as in fast mouse movements.

In pen input, we can make a distinction between those user activities which are the same as in the case of the mouse, i.e., pointing, clicking, and drag/drop operations, and pen movements which represent the exact shape of drawings or handwritten characters.

As a rule, it can be stated that when the pen is 'down', or on the writing surface, the priority of the position samples is high and no sample may be lost (Table 2.3). This

ensures that the handwriting and drawing shapes are recorded faithfully. On the other hand, if the pen is in the air, the priority of the position samples is lower and an occasional drop-out will go unnoticed. As an example, in the Microsoft Windows for Pen Computing environment, the pen replaces the mouse in the usual 'mouse-type' actions which are sent to the MS Windows event queue. However, if the pen goes 'down' in a field in a dialog box where text is written, the system goes into a tight loop, processing the XY coordinates immediately and presenting them as ink on screen without returning to the main event loop after each sample. Similarly, in some pen-based Personal Digital Assistants, the XY-transducer driver immediately displays the ink in a separate bit plane of the graphics device, and buffers the coordinates for processing by the application in the event loop at a lower rate.

Input category	*Priority*
pen-down movement (ink)	no sample loss allowed
pen-up movement	medium priority

Table 2.3: Different sampling priorities in pen interfacing

2.1.3 Sampling modes

There are a number of different sampling modes available in current digitizers. The importance of these modes became apparent during the last decade, such that there is a convergence between different brands of digitizers as regards the definition of each mode. The following modes are typically available:

Continuous sampling (equidistant in time) This mode has two variants:

- ... during both pen-up and pen-down
- ... during pen-down only, with separator records between pen-down streams

Tracking (equidistant in space) Samples are generated when a threshold distance has been traveled with respect to the previous sample. Also called 'mouse mode'.

Pointing Samples are generated when the user taps with the pen on the surface. No live cursor control can be implemented, but for some applications this is not necessary: The user already sees the position pen tip on the surface of a writing pad.

In the latter two modes, time stamps become important for synchronisation and reconstruction of the time axis. This may be necessary in digital filtering, assuming normal time functions.

2.1.4 Inherent feedback

Explanation: Some input modalities inherently require continuous feedback. Examples are force feedback joysticks, but also handwriting on 'Electronic Paper' where the CPU must perform the computations necessary for the inking process. This is opposed to input modalities which have a separate and autonomous sampling process. A given application may require the generation of direct feedback during sampling, but in such a case, the feedback cannot be considered as 'inherently required' by the given input modality.

2.1.5 Channels

Table 2.4 gives an overview of signals which may be recorded with currently available digitizers. Not all commercial systems will provide these given signals.

x, y	position (velocity, acceleration, ...)
p	pen force ("pressure")
	binary pen-up/pen-down switch analog axial force transducer
z	height
ϕ_x, ϕ_y	angles
Switching	by thresholding of pen force p (above) or with additional button(s) on the pen

Table 2.4: Parameters controlled by a pen

2.1.6 Synchronization

What are the provisions that can be taken to make sure that this modality is in synchrony with another relevant modality? In pen computing, the typical solution is to add time stamps to individual samples or to pen-down and pen-up events.

2.1.7 Data formats

Commercial digitizers are provided with a number of data formats, varying from legible ASCII to binary formats. A number of companies has succeeded in creating de facto format standards (Summagraphics, Wacom-II). As a general rule, the legible ASCII format is not

suitable for the required sampling rates. A single coordinate can be represented by a 16 bit number, requiring 5 ASCII characters apart from a separator. There is X, Y, and a switch and/or button signal, requiring 13 or more bytes per sample in legible ASCII. A binary signal can be contained within 7/8 bytes. Usually, the most significant bit of the first byte is set to one, whereas it is zero for the later bytes (see Table 2.5. This allows for an easy synchronization of the data stream. However, it also means that X, Y, switch, pressure and angle coordinates must be extracted from the byte stream by binary shift operations and bit masking.

bit	7	6	5	4	3	2	1	0
byte								
1	1	?	?	?	?	Sx	X15	X14
2	0	X13	X12	X11	X10	X9	X8	X7
3	0	X6	X5	X4	X3	X2	X1	X0
4	0	0	0	0	0	Sy	Y15	Y14
5	0	Y13	Y12	Y11	Y10	Y9	Y8	Y7
6	0	Y6	Y5	Y4	Y3	Y2	Y1	Y0
7	0	Z6	Z5	Z4	Z3	Z2	Z1	Z0

Table 2.5: Example of bit assignment in the bytes received over the serial line from a typical digitizer: Wacom-II format, with pen pressure mode. Note the synchronization bit (MSB) in byte 1, and the sign bits Sx and Sy. To make things complicated, Z is in two's complement.

Examples of simple application file formats are given in Table 2.6. More complex file formats for scientific use (UNIPEN [10]) and for practical applications (Jot [27]) exist.

a) X	Y	P
21	324	1
43	454	1
54	546	1
65	432	0
87	202	0
70	180	0
90	250	1
98	277	1

b) X	Y	T *[ms]*
21	324	3402
43	454	3415
54	546	3419
-1	-1	-1
87	202	3803
70	180	3870
90	250	3875
98	277	3890

Table 2.6: Simple examples of a piece of recorded pen movements. Note the presence of a pen-up stream in the middle of the sequence. a) Continuous time-equidistant sampling at 100 Hz, b) 'Mouse-mode' or tracking mode, pen-down streams only, separated by a pen-up separator record of (-1, -1, -1). In b), samples are necessarily timestamped

.

2.1.8 Software requirements

A serial line driver which handles the *initialization*, the *start* of sampling, the *stop* of sampling, the alignment of the input byte stream and conversion to integer values per channel. Mouse drivers will not suffice mostly, because of their low temporal and spatial resolution. It is essential to know that the users want an immediate inking response.

2.2 Audio

2.2.1 Transducer characteristics & sampling requirements

Parameter	*Min.*	*Max.*	*Typical*	*Units*
Sampling rate	8	48 (good)	22.05/32	kHz
Resolution	8	16 (good)	16	bit
Accuracy	—	—	2^{-Nbits}	bit

Table 2.7: Characteristics of audio sampling devices

Sampling Burst Duration [s]	*Interaction category*
< 1	isolated voice commands
≈ 10	connected-speech sentences
Continuous	(video)phone, music recording

Table 2.8: Different sampling-burst durations in the use of audio input under different application conditions

2.2.2 Sampling priority

The human ear is particularly sensitive to transient disturbances of the audio signal. Pulses and steps in the audio signal must be avoided always. Thus, the audio signal has a very high priority, both in input (A/D) and in output (D/A). In fact, for a number of applications it may be useful to consider the speech sampling clock as the master clock for other sampling processes. It would be unwieldy to use the high sampling rate itself, but audio frames of 10 ms can be synchronized nicely with human movement signals such as from the mouse, joystick or pen.

2.2.3 Sampling modes

(single mode)

2.2.4 Inherent feedback

Sometimes, a recording monitor speaker is needed, but the necessary 'computation' for audio feedback does not have to be part of the application and does not necessarily involve the CPU.

2.2.5 Channels

Mono or Stereo. Examples: 1–2 channels on SUN, 2–4 channels on Indigo/Indy.

2.2.6 Synchronization

Question: What are the provisions that can be taken to make sure that this modality is in synchrony with another relevant modality? On SUN, interrupts are generated at each point in time when a buffer is filled. This yields approximately 8000 samples resolution. On SGI Indigo/Indy the number of sampling frame can be used yielding time in nanoseconds resolution.

2.2.7 Data formats

Raw sampled audio data are usually 8 to 24 bit integer data (PCM 2's-complement) or 32 float float (range -1 . . . 1). Stereo data are usually stored as a sequence of left/right pairs of samples. This data can be transmitted or stored either in this raw format or transformed to formats, which need less bandwidth. Typical data formats use in speech coding are the μ-law coding and A-law coding. These formats reduce the amount of data by applying a logarithmic function to the amplitude in order to compress the dynamic range of the data. On SUN and SGI hardware the conversion to this formats is supported by the audio hardware. MPEG coding of audio data makes use of psychoacoustical masking effects and transmits only the audible parts of the signal. Operations like filtering or mixing only work on raw audio data.

2.2.8 Software requirements

For the recording and playing of raw audio data only a simple driver for the AD/DA hardware is required. Simple audio compression algorithms are implemented on most architectures either in hardware or software. More complex coding schemes like MPEG require additional software modules.

2.3 Single-point Movement and Force Control

If from the user's point of view a single end effector is used, like the human hand holding a single controlling device, this is a case of single-point control. The first extension to this would be bimanual control, as an example. However, the case of single point control applies to many computer applications, more specifically in the control of cursor movement on the screen. Because it is highly undesirable that end applications need to know about all the details of all possible single-point manipulanda (mouse, joystick etc.), a generic software input device is introduced which handles a single time stream of signals: The META DEVICE DRIVER (*MDD*).

The *MDD* is a C-library created by UKA which provides unified access to several hardware input devices, some of which also have the capability to generate haptic (feedback) output. The following description is related to the common denominator of these input devices. A more detailed description will be presented in section 6.1.

2.3.1 Transducer characteristics & sampling requirements

Parameter	*Min.*	*Max.*	*Typical*	*Units*
Sampling rate	—	100	20	Hz
Resolution	*	*	1	1
Accuracy	*	*	1	1

Table 2.9: Characteristics of the output of the META DEVICE DRIVER

In contrast to other modalities/devices, the resolution and the accuracy are not easily specified because the *MDD* covers several different devices with completely different characteristics for input and output. E. g., the values sent by the device (input) vary significantly in their characteristics with respect to the number of dimensions, the range, and the way the input data is generated. The same is true for those devices which can generate haptic output. Therefore—and because the *MDD* is open to new devices which can not be covered here—we decided *not to include* the characteristics of each single device in this section.

For most type of input devices, it is typical that the user has continuous control, and continuous sampling must take place. However, the sampling is not of very high priority, and the occasional loss of a sample will mostly go unnoticed by the user.

2.3.2 Sampling modes

Continuous (equidistant in time), Tracking (equidistant in space), Pointing, Continuous during pen-down, pen-up is a separator sample (and optional time stamp).

2.3.3 Inherent feedback

Some of the input devices will take a default position if they are not manipulated by the user (SpaceMouse, Space- Master, ForceJoystick). The ForceJoystick can also be used in a force feedback mode which may produce continuous force feedback.

2.3.4 Channels

Table 2.10 gives an overview of signals which can be handled by the *MDD*. There are 3 to 7 channels possible.

Device	Translational axes (c/d)	Rotational axes (c/d)	Different button states	Haptic output?
ForceMouse	2/0	0/0	4	vibration/brake
ForceJoystick	2/0	0/0	8	force/position
CyberMan	2/1	0/3	8	vibration
SpaceMaster	3/0	3/0	2	No
SpaceMouse	3/0	3/0	512	No

Table 2.10: Parameters controlled by different I/O devices (c = continuous, d = discrete)

2.3.5 Synchronization

What are the provisions that can be taken to make sure that this modality is in synchrony with another relevant modality? Although not supported at the moment, time stamps may be added to the data samples transmitted from the specific device driver to the meta device driver easily.

2.3.6 Data format

The basic idea of the META DEVICE DRIVER is to provide unified access via a general interface to various devices. Therefore, the *MDD* provides functions for reading input values and sending control commands, and it provides a general data structure managed by the specific device drivers.

The following structure is intended to store device data, i. e. values send by the device. As described before, the position, the orientation, and the button state have been implemented. If a device supports less than six degrees of freedom, the unused values will be set to '0'.

```
typedef struct {
   int tx,ty,tz;          translational values
   int rx,ry,rz;          rotational values
   int btnState;          button states
} deviceData;
```

2.3.7 Software requirements

The *MDD* has been realized as an independent C-library which can be used by many applications. Its implementation is based on software which is available for free only. In its current version, the *MDD* needs a UNIX environment, a C and a C++ compiler, the communication tool PVM (see 1.1.5), and the GUI toolkit Tcl/Tk (see 1.1.2). It comes with configure scripts and independent Makefiles and has been tested on SunOS, Solaris, and Irix.

2.4 Multiple-point Control with the Exoskeleton

As a natural extension to single-point control, where a single human end effector produces signals along a number of dimensions, in multiple-point control the user may also simultaneously produce signals along a number of dimensions, but now for several end effectors at the same time. The exoskeleton (EXO) from DIST allows the acquisition of the angular variables of a multi-joint exoskeleton structure with one or two arms to be transmitted to a forward kinematic module which can reconstruct position/orientation of the end-effector of the structure. At each joint, there is a low-friction potentiometer.

2.4.1 Transducer characteristics & Sampling requirements

Parameter	*Min.*	*Max.*	*Typical*	*Units*
Sampling rate	10	50	—	Hz
Resolution	—	—	—	rad
Accuracy	—	—	—	rad

Table 2.11: Characteristics of exoskeleton joint-angle sensor devices

Sampling Burst Duration	*Interaction category*
Continuous	Whole-body movement

Table 2.12: Sampling time window

2.4.2 Inherent feedback

In many applications the parameters derived from the joint angles will be fed to a feedback system, on line. For example: movement leads to acoustical parameter changes, or movement leads to graphical humanoid movement on screen. As such, this feedback is not inherent to the device, but the applications are usually highly interactive, requiring immediate feedback.

2.4.3 Channels

θ_i	six angles per arm, 1 or 2 arms

Table 2.13: Parameters controlled by the exoskeleton

2.4.4 Data formats

(an internal data format is being used)

2.4.5 Software/Hardware Requirements

Software: Winsocket library

Hardware: PC with A/D converter and multi-channel multiplexer

Real-Time: real-time is necessary. Tests showed a better performance on Win95 than on Windows NT.

Chapter 3

Representation of Input and Output Channels

This chapter is about the representation of input and output channels. The natural modalities are audio, video, handwriting and single point movements with feedback. Interdependencies between modalities are discussed. The description of the different channels leads to a basic architecture for multimodal/multimedia application.

This chapter starts with a general overview of all input and output modules which are involved. After introducing a classification scheme each module is presented in a schematic way. It turns out the modules are dealing with different levels in the processing hierarchy (from hardware level to high level symbolic processing). The complete architecture for input and output handling is presented reflecting the different levels in the processing hierarchy. Where possible examples of bimodal integration are shown. As processing power is limited it might help to give each of the modalities appropriate priorities to ensure an effective and plausible interaction with the user. Some aspects for assigning priorities are discussed. Furthermore mechanisms the synchronization have to provided. The criteria for synchronization and the techniques, which are implemented in MIAMI are presented. For a detailed discussion of the implementation of the modules please refer to the software documentation and to the presentation in chapters 6 and 7.

3.1 Overview of Modules for Input and Output Channels

This chapter is based on a formal description of the following modules designed in MIAMI:

HARP-MIDIKER	MIDI output handling
MDD	Meta Device Diver
AAS	Audio Application Server
ICP-FACE-ANIM	Face Animation
BIMODAL-ICP-TTS	Bimodal Text to Speech System
EXO	Filtering Exoskeleton Data
HARP-VSCOPE	Real-Time Tracking and Recognition of body gestures
ICP-LIP-METER	Extraction of Lip Parameters
BIMODAL-ICP-ASR	Bimodal Automatic Speech Recognizer

As introduced in chapter 2, the different modules which are handling the different sensors or effectors can by described by their:

- Function
- Input signals
- Output signals
- Parameters in (to change the mode of the module)
- Parameters out (to inform other modules about status)

For input and output signals and the parameters the sampling or transmission rates, as well as the data formats and the contents are presented. Also the way of module intercommunication (synchronous/asynchronous) and whether the the module is permanently active or only sending/receiving on request is shown. If possible, the sender or receiver of the different modules are identified and other modules, which have to be synchronized to this stream are listed.

In the following sections each module is presented in detail according to the scheme introduced above.

3.2 Midi Mapper (*HARP-MIDIKER*)

HARP-MIDIKER allows to map multiple MIDI-inputs on a single MIDI-output device in real time.

3.2.1 Requirements

Software:	Windows 95
Hardware:	MPU Roland 401 compatible MIDI interface
Real-Time:	necessary

3.2.2 Data channels

Input

Number of channels:	1 to 10
Description:	a MIDI packet containing the MIDI data and the time stamp
Type of sending agent:	music generating software agents
Format of input data:	pairs of unsigned integers
Transmission rate:	as requested from the music agent
Synchronous/asynchronous data input:	synchronous
Synchronization with other modules:	—
continuous/on request:	on request
Priority:	high

Output

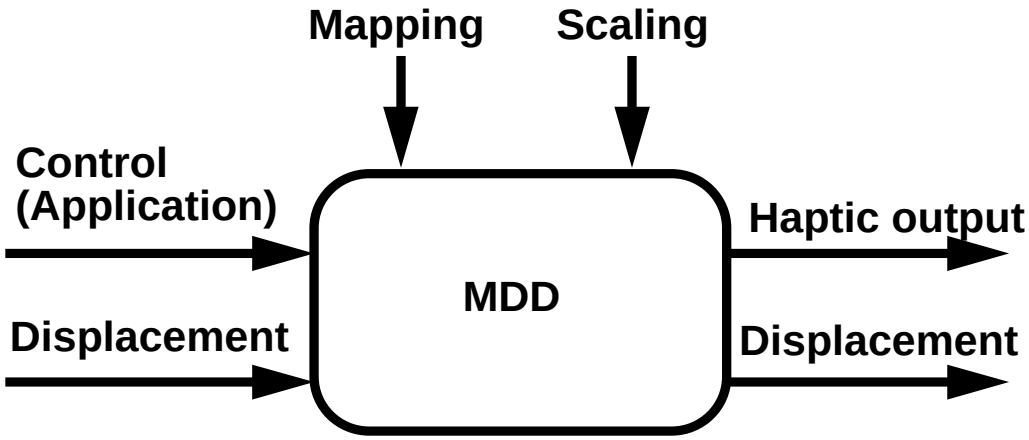
Number of channels:	1
Description:	standard MIDI
Receiving agent:	Win32 multimedia kernel
Format of output data:	standard MIDI data stream
Transmission rate:	resolution 1 ms
Synchronous/asynchronous data output:	asynchronous
Synchronization with other modules:	-
Continuous/on request:	continuous
Priority:	high

3.2.3 Control parameter

No control parameter.

3.3 Meta Device Driver (*MDD*)

The *MDD* is a C-library which provides unified access to several input devices, some of which also have the capability to generate haptic output. A more detailed description will be presented in section 6.1.



3.3.1 Requirements

Software:	any UNIX-like operating system
Hardware:	Implemented and tested on SUN and SGI workstations
Real-Time:	More or less possible and necessary for direct manipulation

3.3.2 Data channels

Input

Number of channels: 2

Channel 1	Description:	control commands
	Type of sending agent:	application program
	Format of input data:	— (no common format)
	Transmission rate:	selectable
	Synchronous/asynchronous data input:	asynchronous
	Synchronization with other modules:	no
	continuous/on request:	initiated by application program
	Priority:	low
Channel 2	Description:	displacement values from an input device
	Type of sending agent:	device driver
	Format of input data:	6 integers representing displacement values, 1 integer representing the button state
	Transmission rate:	depends on specific driver (10–100 Hz)
	Synchronous/asynchronous data input:	asynchronous
	Synchronization with other modules:	handshake between SDD and *MDD*
	continuous/on request:	depends on specific driver
	Priority:	high

Output

Number of channels: 2

Channe1	Description:	haptic output functions
	Receiving agent:	device driver
	Format of output data:	depends on specific driver
	Transmission rate:	unknown, single events only
	Synchronous/asynchronous data output:	asynchronous
	Synchronization with other modules:	handshake between SDD and *MDD*
	Continuous/on request:	on request
	Priority:	high

Channe2	Description:	input values for direct manipulation
	Receiving agent:	application program
	Format of output data:	6 integers representing displacement values, 1 integer representing the button state
	Transmission rate:	not fixed (10–100 Hz)
	Synchronous/asynchronous data output:	asynchronous
	Synchronization with other modules:	—
	Continuous/on request:	continuous
	Priority:	high

3.3.3 Control parameter

In

Number of parameters: 12 parameters for mapping, 6 parameters for scaling

Mapping	Description:	mapping a physical axis/degree of freedom to a logical axis/degree of freedom
	Format:	2 constants (phys, log)
	Transmission rate:	single events
	Will be changed by:	application program, options menu

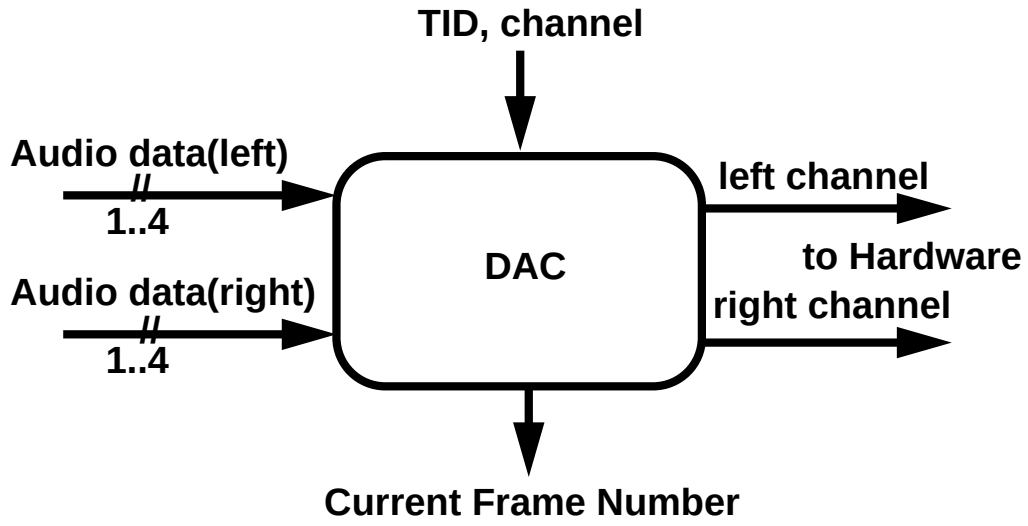
Scaling	Description:	scaling a logical axis/degree of freedom
	Format:	constant combination ('mask'), 1 double (scaling factor)
	Transmission rate:	single events
	Will be changed by:	application program, options menu

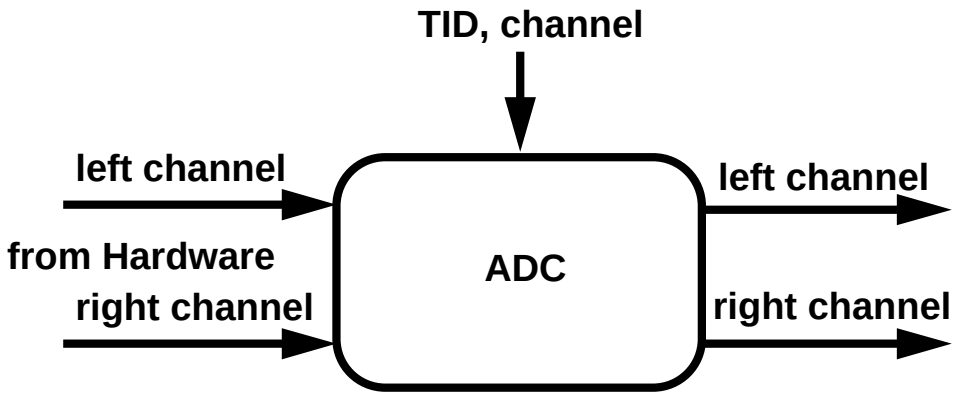
Out

Same as In parameters.

3.4 ADC and DAC

ADC (Analog Digital Converter) and DAC (Digital Analog Converter) allow a unified access to the audio hardware of Sun SPARCstation and SGI Indigo/Indy. Multiple applications can read and write frame the hardware at the same time. DAC also broadcasts a synchronization signal to all task, which are members of the "sync"-group.





3.4.1 Requirements

Software:	SunOS, Sun Solaris2.4, IRIX 5.3, PVM 3
Hardware:	SUN SPARC, Indigo, Indy
Real-Time:	necessary

3.4.2 Data channels for DAC

Input

Number of channels:	minimum 1 (depends on application and processing power)
Description:	audio data
Type of sending agent:	any other application
Format of input data:	array with 16 bit integer values
Transmission rate:	selectable (8 kHz–44.1 kHz, typically 22.05 kHz)
Synchronous/asynchronous data input:	synchronous
Synchronization with other modules:	*ICP-FACE-ANIM*
continuous/on request:	continuous and on request
Priority:	high

Output

Number of channels:	1–2 (on SUN) 1–4 on (Indigo/Indy)
Description:	Audio signal
Receiving agent:	DAC
Format of output data:	Integer (16 bit)
Transmission rate:	same as input rate
Synchronous/asynchronous data output:	synchronous
Synchronization with other modules:	*ICP-FACE-ANIM*
Continuous/on request:	continuous
Priority:	high

3.4.3 Data channels for ADC

Input

Number of channels:	minimum 1
Description:	raw audio data
Type of sending agent:	hardware driver
Format of input data:	array with 16 bit integer values
Transmission rate:	selectable (8 kHz–44.1 kHz, typically 22.05 kHz)
Synchronous/asynchronous data input:	asynchronous
Synchronization with other modules:	—
continuous/on request:	continuous and on request
Priority:	high

Output

Number of channels:	1–2 (on SUN) 1–4 on (Indigo/Indy)
Description:	Audio signal
Receiving agent:	DAC
Format of output data:	Integer (16 bit)
Transmission rate:	same as input rate
Synchronous/asynchronous data output:	synchronous
Synchronization with other modules:	*BIMODAL-ICP-ASR*
Continuous/on request:	continuous
Priority:	high

3.4.4 Control parameter

In

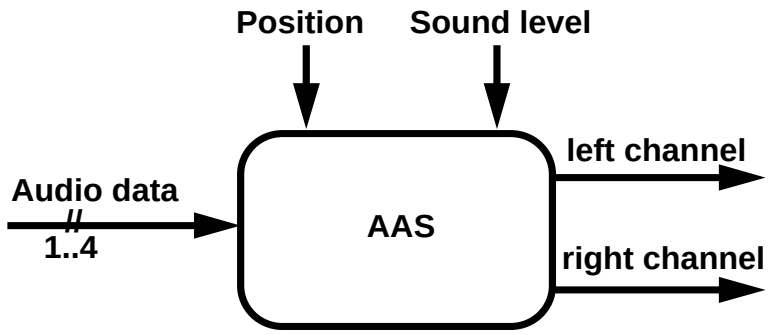
Number of parameters:	1
Description:	Connect new sender/receiver
Format:	2 integer, task ID and channel of sender/receiver
Transmission rate:	—
Will be changed by:	any client

Out

Number of parameters:	1
Description:	current frame number
Format:	long integer 32 bit
Transmission rate:	25 Hz
Will change the status or inform:	*ICP-FACE-ANIM*, Audioscript

3.5 Audio Application Server (*AAS*)

AAS (Audio Application Server) handles multiple input and output sound streams. The generation of simple audio signals and the real-time spatialization of sound sources is performed.



3.5.1 Requirements

Software:	SunOS, Sun Solaris2.4, IRIX 5.3, PVM 3
Hardware:	SUN SPARC, Indigo, Indy
Real-Time:	necessary

3.5.2 Data channels

Input

Number of channels:	minimum 1 (depending on application and processing power)
Description:	audio data
Type of sending agent:	any other application
Format of input data:	array with 16 bit integer values
Transmission rate:	selectable (8 kHz–44.1 kHz, typically 22.05 kHz)
Synchronous/asynchronous data input:	synchronous
Synchronization with other modules:	*ICP-FACE-ANIM*
continuous/on request:	continuous and on request
Priority:	high

Output

Number of channels:	1–2 (on SUN) 1–4 on (Indigo/Indy)
Description:	Audio signal
Receiving agent:	DAC
Format of output data:	Integer (16 bit)
Transmission rate:	same as input rate
Synchronous/asynchronous data output:	synchronous
Synchronization with other modules:	*ICP-FACE-ANIM*
Continuous/on request:	continuous
Priority:	high

3.5.3 Control parameter

In

Number of parameters: 2

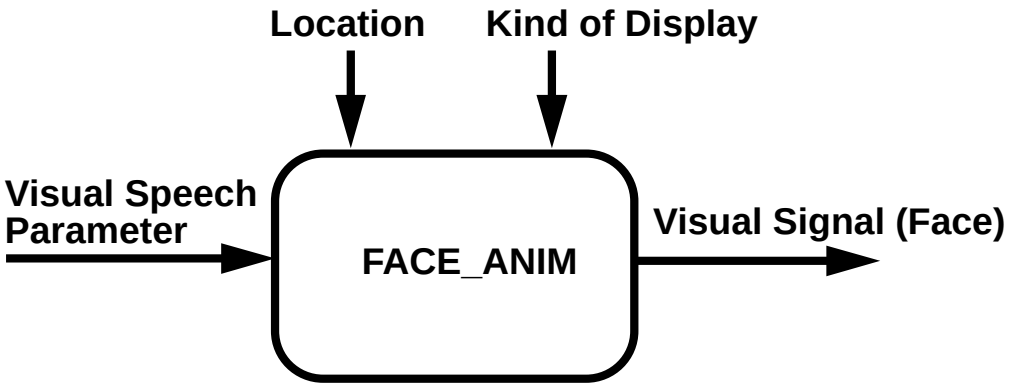
Parameter 1	Description:	Position of Sound Source
	Format:	3 Integer (Elevation, Azimuth and Source ID)
	Transmission rate:	max. 25 Hz
	Will be changed by:	any client
Parameter 2	Description:	Sound level
	Format:	2 Integer (Level and Source ID)
	Transmission rate:	max. 25 Hz
	Will be changed by:	any other client

Out

No parameter.

3.6 Face Animator (*ICP-FACE-ANIM*)

Module *ICP-FACE-ANIM* animates a 3D-face or part of a face on the screen.



3.6.1 Requirements

Software:	IRIX 5.3, OpenGL, pvm, TCL/TK
Hardware:	SGI
Real-Time:	necessary

3.6.2 Data channels

Input

Number of channels:	1
Description:	Visual Speech Parameter
Type of sending agent:	*BIMODAL-ICP-TTS*, *ICP-LIP-METER*
Format of input data:	array of int
Transmission rate:	25 Hz
Synchronous/asynchronous data input:	asynchronous
Synchronization with other modules:	*BIMODAL-ICP-TTS*, *ICP-LIP-METER*
continuous/on request:	—
Priority:	high

Output

Number of channels:	1
Description:	Visual Signal
Receiving agent:	Screen
Format of output data:	Integer (16 bit)
Transmission rate:	25 Hz
Synchronous/asynchronous data output:	—
Synchronization with other modules:	DAC
Continuous/on request:	continuous
Priority:	high

Channel 2

3.6.3 Control parameter

In

Number of parameters: 2

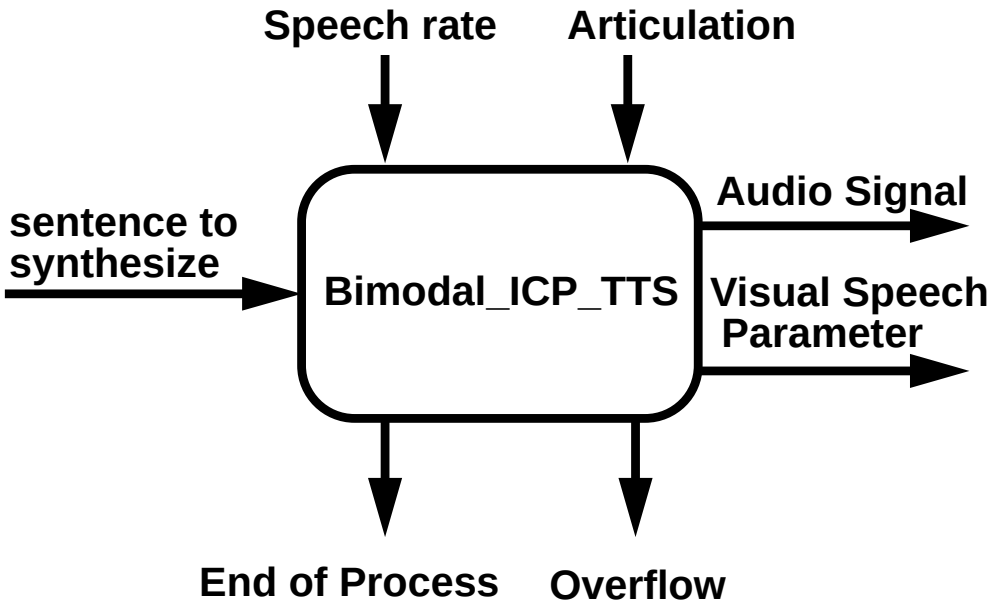
Parameter 1	Description:	Location
	Format:	array of 6 Integer
	Transmission rate:	
	Will be changed by:	6D Mouse, *MDD*
Parameter 2	Description:	kind of display
	Format:	int
	Transmission rate:	—
	Will be changed by:	—

Out

No parameter.

3.7 Text to bimodal speech (*BIMODAL-ICP-TTS*)

BIMODAL-ICP-TTS is a bimodal text to speech system. It converts ASCII text into audiovisual speech.



3.7.1 Requirements

Software:	IRIX 5.3, PVM 3
Hardware:	SGI
Real-Time:	not possible

3.7.2 Data channels

Input

Number of channels:	1
Description:	sentence to synthesize
Type of sending agent:	Handwriting Recognition, *BIMODAL-ICP-ASR*
Format of input data:	ASCII Text
Transmission rate:	selectable (10–50 Hz)
Synchronous/asynchronous data input:	asynchronous
Synchronization with other modules:	—
continuous/on request:	—
Priority:	medium

Output

Number of channels: 2

Channel 1	Description:	Audio signal
	Receiving agent:	Audio server (PAC)
	Format of output data:	Integer (16 bit)
	Transmission rate:	44.1 kHz
	Synchronous/asynchronous data output:	asynchronous
	Synchronization with other modules:	Audio server
	Continuous/on request:	continuous
	Priority:	high

Channel 2	Description:	Visual Speech Parameter
	Receiving agent:	*ICP-FACE-ANIM* (animated face)
	Format of output data:	array of int
	Transmission rate:	25 Hz
	Synchronous/asynchronous data output:	asynchronous
	Synchronization with other modules:	*ICP-FACE-ANIM*
	Continuous/on request:	continuous
	Priority:	high

3.7.3 Control parameter

In

Number of parameters: 2

Parameter 1	Description:	Speech Rate
	Format:	Integer
	Transmission rate:	—
	Will be changed by:	—
Parameter 2	Description:	Articulation
	Format:	Integer
	Transmission rate:	—
	Will be changed by:	—

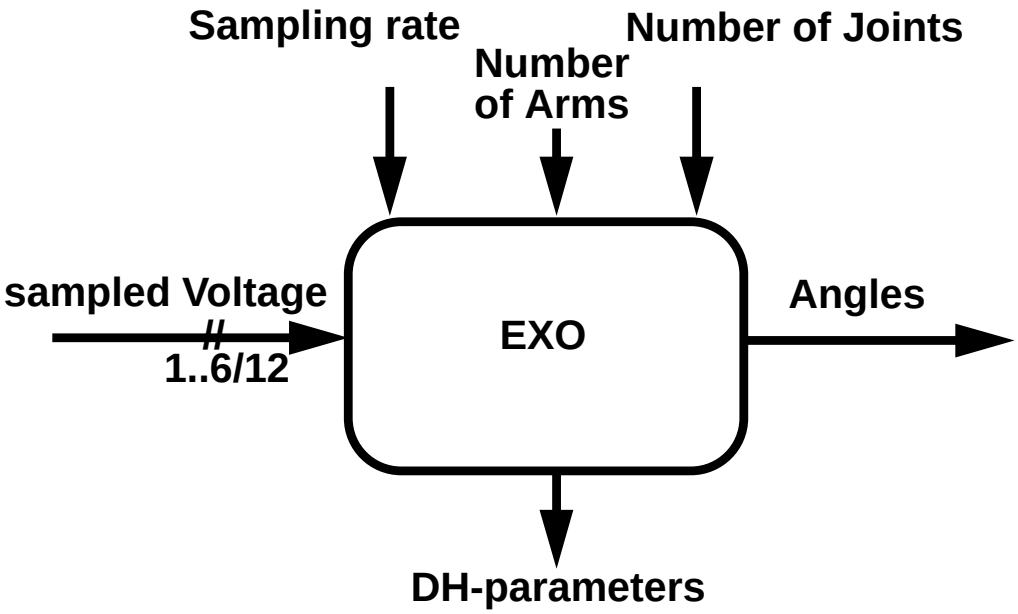
Out

Number of parameters: 2

Parameter 1	Description:	End of Process
	Format:	Integer
	Transmission rate:	—
	Will change the status or inform:	*ICP-FACE-ANIM*, Audio server
Parameter 2	Description:	Overflow
	Format:	Integer
	Transmission rate:	—
	Will change the status or inform:	Handwriting Recognition *BIMODAL-ICP-ASR*

3.8 Exoskeleton module (*EXO*)

EXO allows the acquisition of the angular variables of a multi-joint exoskeleton structure with one or two arms to be transmitted to a forward kinematic module, which can reconstruct the position or orientation of the end-effector of the structure.



3.8.1 Requirements

Software:	Winsocket library
Hardware:	PC with A/D converter and multi-channel multiplexer
Real-Time:	necessary

3.8.2 Data channels

Input

Number of channels:	6 (one arm) or 12 (two arms) all with same data format
Description:	Voltage proportional to angles of joints
Sending agent:	D/A converter
Format of input data:	analog voltage signal
Transmission rate:	selectable (10-50 Hz)
Synchronous/asynchronous data input:	synchronous
Synchronization with other modules:	—
continuous/on request:	continuous
Priority:	high

Output

Number of channels:	1 or 2, all with same data format
Description:	angles of joints
Receiving agent:	forward kinematic module
Format of output data:	6-dimensional vector of reals
Transmission rate:	selectable (10-50 Hz)
Synchronous/asynchronous data output:	synchronous
Synchronization with other modules:	—
Continuous/on request:	on request
Priority:	high

3.8.3 Control parameter

In

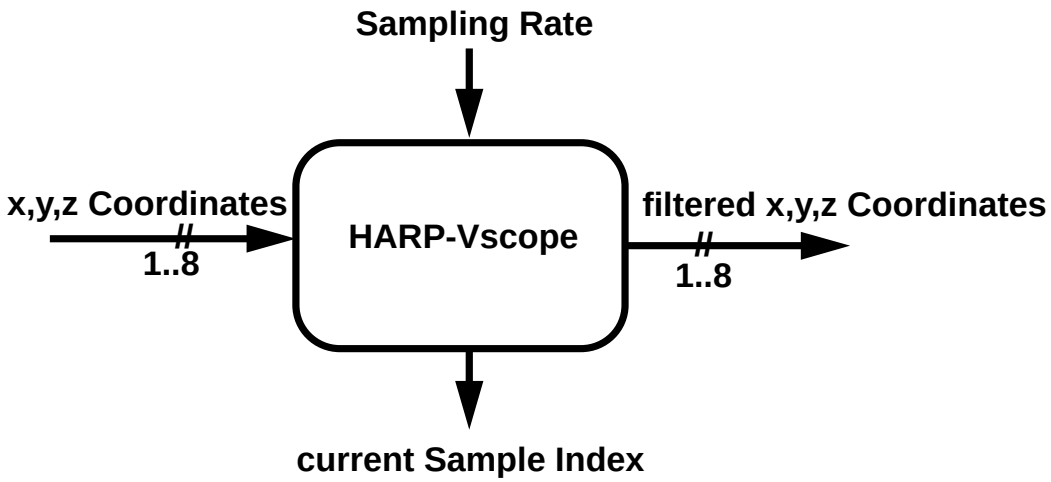
Number of parameters:	3
Description:	sampling rate (S), number of arms (NA), number of joints (NJ)
Format:	3 integers
Transmission rate:	Information is only received during startup
Will be changed by:	exoskeleton configuration module

Out

Number of parameters:	4 x number of joints (NJ)
Description:	Denavit-Hartenberg parameters of each joint
Format:	4 reals (2 distances + 2 angles)
Transmission rate:	Information is only sent during startup
Will change the status or inform:	forward kinematic module

3.9 Full-body gesture module (*HARP-VSCOPE*)

HARP-Vscope is a real-time tracking and recognition system of full-body gestures via infrared and ultrasound sensors based on the V-scope hardware.



3.9.1 Requirements

Software:	Windows 95
Hardware:	V-scope
Real-Time:	necessary

3.9.2 Data channels

Input

Number of channels:	1 to 8, all with same data format
Description:	x, y, and z coordinates of trajectory points
Type of sending agent:	V-scope
Format of input data:	triples of unsigned integers
Transmission rate:	10 ms to 80 ms for each channel
Synchronous/asynchronous data input:	asynchronous
Synchronization with other modules:	—
continuous/on request:	continuous
Priority:	high

Output

Number of channels:	1 to 8 channels, all with same data format
Description:	filtered (x, y, z) coordinates (in meters)
Type of receiving agent:	sound output agent; humanoid animation agent
Format of output data:	buffer of the last 256 triples for each channel
Transmission rate:	10 ms to 80 ms for each input channel
Synchronous/asynchronous data output:	synchronous
Synchronization with other modules:	—
Continuous/on request:	on request
Priority:	medium

3.9.3 Control parameter

In

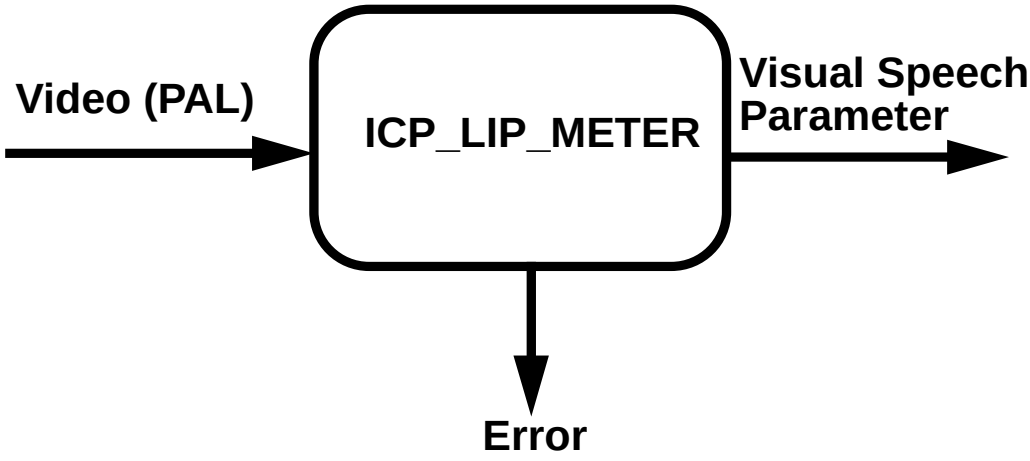
Number of parameters:	1
Description:	sampling rate
Format:	integer
Transmission rate:	low
Will be changed by:	high level task supervisor

Out

Number of parameters:	1
Description:	current sample index
Format:	integer
Transmission rate:	on demand
Will change the status or inform:	client agents

3.10 Lip Parameter Analyzer (*ICP-LIP-METER*)

Module *ICP-LIP-METER* extracts the lip parameter from a video signal of a human speaker with lips made up in blue.



3.10.1 Requirements

Software:	IRIX 5.3 , Video Library vl (VINO card), Library gl
Hardware:	Indy , video card VINO
Real-Time:	necessary, possible

3.10.2 Data channels

Input

Number of channels:	1
Description:	Video PAL signal
Type of sending agent:	VCR, Camera
Format of input data:	—
Transmission rate:	selectable 25 frames/sec
Synchronous/asynchronous data input:	asynchronous
Synchronization with other modules:	—
continuous/on request:	—
Priority:	high

Output

Number of channels:	1
Description:	Visual Speech Parameters
Receiving agent:	*ICP-FACE-ANIM*, *BIMODAL-ICP-ASR*
Format of output data:	Array of Integer (16 bit)
Transmission rate:	25 Hz
Synchronous/asynchronous data output:	—
Synchronization with other modules:	—
Continuous/on request:	—-
Priority:	high

3.10.3 Control parameter

In

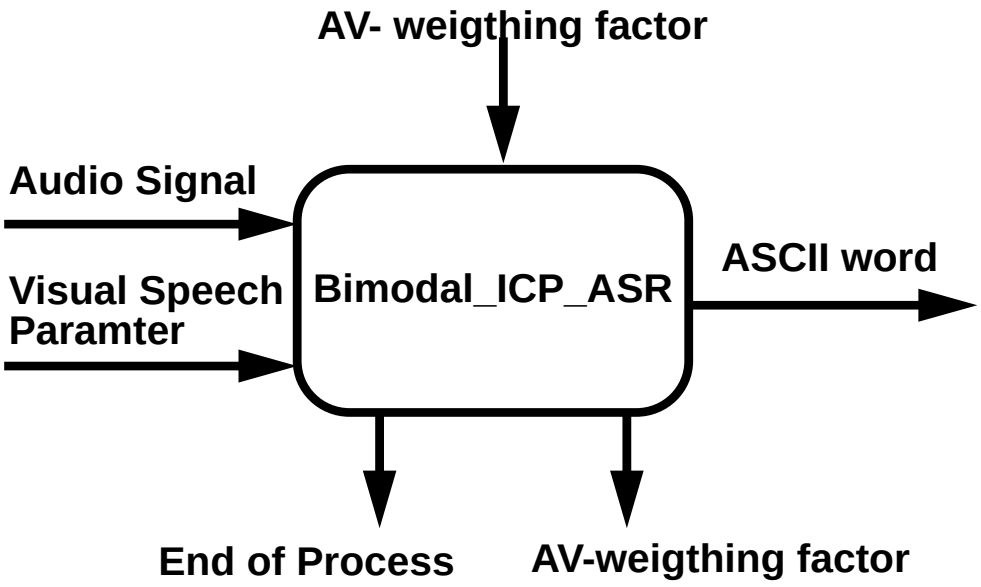
No parameter.

Out

Number of parameters:	1
Description:	bit for measurement errors
Format:	Integer
Transmission rate:	—
Will change the status or inform:	*BIMODAL-ICP-ASR*, *ICP-FACE-ANIM*

3.11 Bimodal speech recognizer (*BIMODAL-ICP-ASR*)

BIMODAL-ICP-ASR (Automatic Speech Recognizer) audio and video input for bimodal speech recognition.



3.11.1 Requirements

Software:	UNIX, LINUX, HMMICP library
Hardware:	—
Real-Time:	not already implemented but possible

3.11.2 Data channels

Input

Number of channels:	2

Channel 1	Description:	Audio signal
	Type of sending agent:	ADC
	Format of input data:	Integer
	Transmission rate:	8 kHz–44.1 kHz
	Synchronous/asynchronous data input:	synchronous
	Synchronization with other modules:	ADC/DAC
	continuous/on request:	continuous
	Priority:	—

Channel 2	Description:	Visual speech parameter
	Type of sending agent:	ICPLipMeter
	Format of input data:	array of float
	Transmission rate:	can be configured (9600 baud on initialization)
	Synchronous/asynchronous data input:	serial asynchronous input
	Synchronization with other modules:	ADC/DAC
	continuous/on request:	
	Priority:	medium

Output

Number of channels:	1
Description:	ASCII word
Receiving agent:	e.g., *BIMODAL-ICP-TTS*
Format of output data:	ASCII
Transmission rate:	—
Synchronous/asynchronous data output:	—
Synchronization with other modules:	—
Continuous/on request:	—
Priority:	—

3.11.3 Control parameter

In	Number of parameters:	1
	Description:	Input Audio/Video weighting factor
	Format:	float
	Transmission rate:	—
	Will be changed by:	*ICP-LIP-METER*

Out Number of parameters: 2

Parameter 1	Description:	End of Process
	Format:	Integer
	Transmission rate:	—
	Will change the status or inform:	*ICP-FACE-ANIM*, Audio server
Parameter 2	Description:	Audio/Video weighting factor based on output probability dispersion
	Format:	float
	Transmission rate:	—
	Will change the status or inform:	—

3.12 Conclusions

3.12.1 Hierarchy of modules

For the input and output channels the following four hierarchies are identified.

- External hardware
- Hardware and operating system dependent driver
- Hardware independent driver
- High level processing

They will be explained in more detail in the following paragraphs.

External Hardware refers to the sensors (for input) and the effector (for output).

Hardware and operating system dependent drivers are firmware, like the driver for the audio or video hardware on a specific architecture.

Hardware independent drivers In some cases partners developed hardware independent drivers to allow a unified access through the same interface to different devices (like the Meta Device Driver) for Mouse, Joystick etc. or the DAC (Digital Analog Converter Driver) and ADC (Analog Digital Converter Driver) for audio devices) or to allow extra buffering and filtering of the raw sampled data (e.g. EXO, HARP-MIDIKER).

High level processing In the class of high level processing two types of modules can be found. One group consists of simple unimodal modules, which only have one input and one output modality. The other contains modules, which are connected to two modalities (audio and video). In the case of the Text-to-Speech system, the level of the output channels is also different. The module produces raw audio data and parameter for the face animation. The first channel can be played back without any further processing, but the second channel needs further processing through the *ICP-FACE-ANIM* module.

The diagrams 3.1 and 3.2 show the links between the different modules and their position in the processing hierarchy. It is obvious from this diagrams that the audio and video input and output have the closest links. All other channels do not show close links on the levels which are presented.

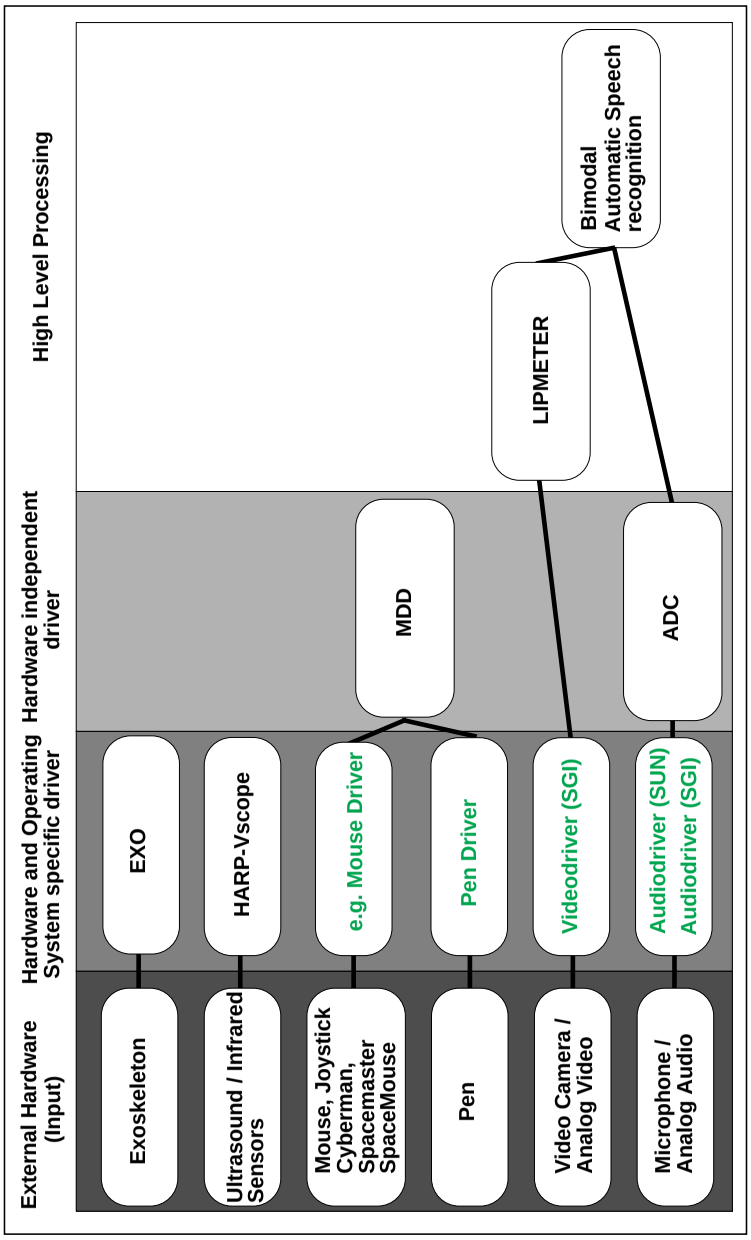


Figure 3.1: Overview of *Input Channels*. The modules with gray text are firmware. All other software modules are developed in the MIAMI-project.

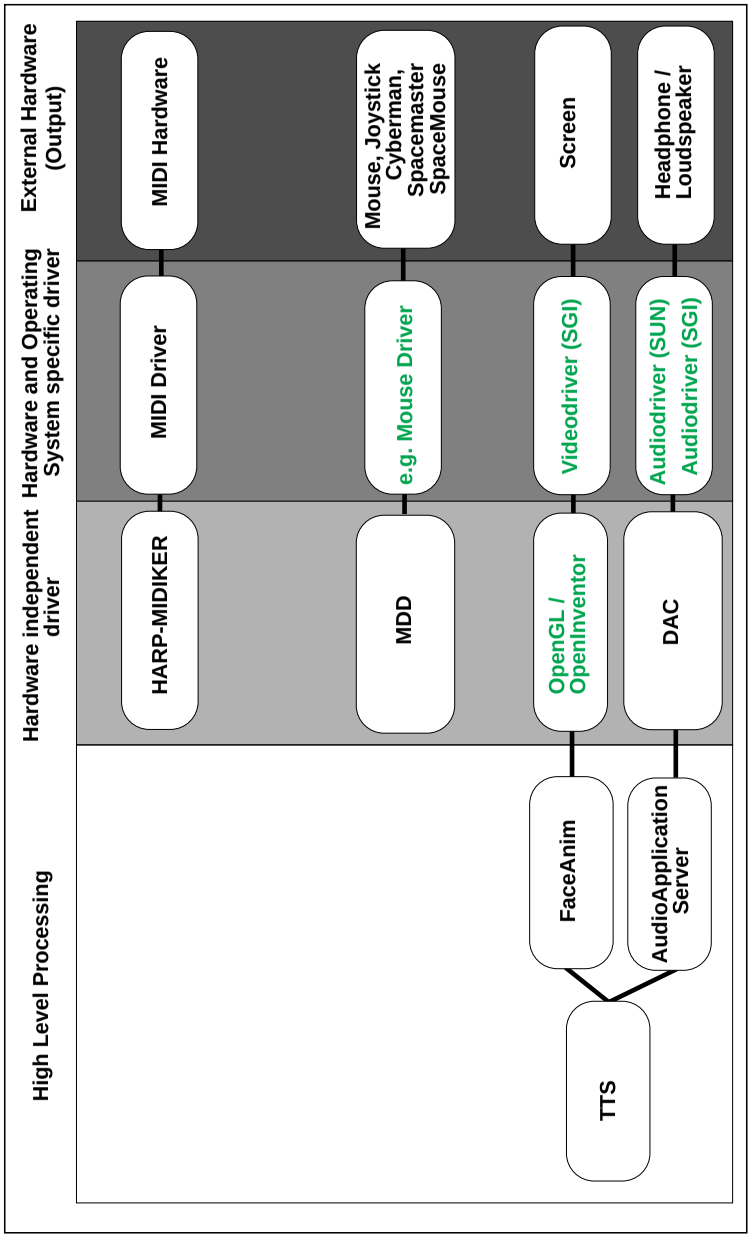


Figure 3.2: Overview of *Output Channels*. The modules with gray text are firmware. All other software modules are developed in the MIAMI-project.

3.12.2 Sampling rate and priorities

The overview of the the different input and output channels can be summarized in the following way. With regards to the sampling rates, three groups can be identified.

- high sampling rates (> 22 kHz): Audio in/out
- medium sampling rates (50–200 Hz): Pen
- low sampling rates (10–50 Hz): Video in/out, mouse, joystick, haptic feedback, body gestures

Audio does not only have the highest sampling rate but also the highest priority, because any lost or delayed sample will be perceived as disturbing noise or decrease the speech recognition rate of the user and of the automatic recognizer. The MIDI channel also request for a high priority with relatively high transmission rates. A delayed note will change the rhythm of a pattern noticeable. The next important channel is the pen input channel. For correct recognition and for a smooth reproduction of shapes, a high sampling rate with no breaks is necessary. The pen channel is followed by video input and output. The preliminary experiments with the animated face (see Deliverable 1, chapter 4) also showed, that a lost frame during the output does not distort the recognition of audiovisual speech significantly. The other channels seem to be represented sufficiently by low sampling rates and are not very sensitive to interrupts.

3.12.3 Interprocess communication

Most of the presented modules are independent tasks, which run on a UNIX multiprocessor environment. For interprocess communication PVM-messages are used. All used data formats are supported by PVM. This also allows a fast adaptation to different hardware platforms. For tasks running under Windows/Windows95/WindowsNT a PVM gateway is available.

3.12.4 Synchronization

As the audio channel has the highest priority and is active most of the time this channel will synchronize the other. As the sampling of sound is controlled by a stabilized oscillator the audio channel also gives stable time base. The unit for the synchronization is called a frame. The duration of a frame ranges from 25 Hz to 200 Hz depending on the demands of the application. The audio samples are stored in an intermediate buffer, which has

duration between 5 ms (200 Hz sampling rate for pen) and 40 ms (25 Hz sampling rate for pen). When the buffer is filled the current frame number is increased and broadcasted to other channels (via PVM-messages). If the processor is fast enough tight synchronization with the resolution of one frame is possible. In cases, when the different channels are requesting more calculation time than available, the modules with less priority will switch to loose synchronization. If such a module notices, that it is too late to transmit the requested data in time, it will skip some frames and continue to process the current frame. For example, if the module for the face animation is too slow to deliver a frame synchronously to the audio stream, it will skip the parameters of the next frame, which the text to speech system transmitted and animate the following parameters instead. In applications where no audio is requested a software based pseudo-clock will count the frame number and broadcast it.

3.12.5 Problems

The sections above showed that some modalities have very tight restrictions with regards to sampling rate and interrupts. Experiments with some bimodal applications or even single audio applications on SUN and SGI platforms demonstrated that they cannot fulfill the tight conditions under all circumstances. Of course, limited processing power often causes some problems if audio and graphical applications are active at the same time. But also with sufficient processing power, the operating system UNIX is not optimal for multimedia applications. Some daemons residing in the background, periodically start their work interrupting the handling of input and output channels. In the moment there are no means to stop these unpredictable interrupts, but maybe further improvements of the operating system and increasing processing power will help to solve these problems.

3.12.6 Summary

In this chapter we gave an overview of the different input and output channels. The different levels which are involved in handling input and output have been identified. Based on the sampling rate and type of transmission priorities have been assigned to the different tasks (Audio, Pen, Video, tactile, haptic). A model for synchronization of the different modalities has been proposed and implemented.

Chapter 4

Representation of Object Space

The representation of object space together with the representation of input and output channels (see chapters 2 and 3) establishes a framework for the development of multimodal systems. Therefore, the main part of this chapter deals with the internal representation of objects, including object properties, data storage, and management. Since the representation of output channels has already been discussed in chapter 3, we here focus on how visual, acoustical, and haptic Inputs/Outputs are coherently processed in synchrony.

4.1 Introduction

Interactions with real and virtual objects are based mainly on three fundamentals: input (see chapter 2), output (see chapter 3), and internal representation of object space. As for the previous two chapters, the visual modality ('Computer Graphics') is well-known and a great number of systems and libraries exist for creating, managing, and displaying objects, e. g., OpenGL and OpenInventor, which have been adopted for the project (see sections 1.1.3 and 1.1.4). In those last two examples, only visual properties of objects are supported, whereas acoustical and haptic properties are not.

Therefore, we defined general MIAMI object properties suitable for the synchronization of the multimodal objects that we have worked on. A presentation of each MIAMI module has been given in the previous section 3. We will here describe how these modules are stored on the computer and how they can be combined together to create a "multimodal object". The MIAMI approach will be illustrated through four examples worked out in collaboration. Since these examples will be presented as "demonstrations" at the Bochum meeting in March, this chapter also aims at serving as a technical description of these demos.

We have organized this section according to the following structure. We first present the general data formats of the MIAMI objects. Detailed description of the graphic, acoustic and haptic representations are then given. The types of display used to present objects to the user are presented too. Finally, the multimodal integration of the MIAMI objects are illustrated in the light of four demonstrations worked out by the MIAMI partners.

4.2 Graphical Representation

One of the major contributions of the MIAMI project has been to agree on and standardize data formats which allow any object worked out by a MIAMI partner to be integrated in a multimedia MIAMI environment. Of course, their integration to other multimodal spaces is not limited to the MIAMI platform since the properties below detailed are mostly based on standardized or widely used references. The common properties of these objects are described in this section, depending on the modality they are to be presented through to the user.

4.2.1 Fundamentals of Computer Graphics

The most salient graphical properties of the MIAMI objects were based on the Graphic Libraries adopted by the partners to serve as software standard, namely OpenGL and OpenInventor. Anyway, whatever the graphic library (and the hardware) used, the way objects are described and stored on the computer is highly similar if not identical.

Defining and storing objects

One of the first problem of Computer Graphics is to store the volume of an object. Since the whole volume of the object cannot be stored (which is by definition infinite), the object as to be approximated by a new volume defined by a finite number of data.

One technique, widely use in CAD/CAM applications, is to define an object as a combination of simple objects. Then, from a set of predefined objects (cube, sphere, cylinder, cone, ...) and by using a set of boolean operations such as union, intersection and difference, a complex object can be built and stored in the computer. However, this technique can only be used for manufactured objects.

Another technique used is to sample the surface of the object and then approximate it with polygon meshes. An object is then defined by geometrical information (a list of all the vertices) and by topological information (the way how the different vertices are

connected together). Although this technique is widely used, some objects can need a smoother approximation of their surface. One solution is to oversample the surface of the object and then use more vertices. A better technique is to use parametric surfaces.

Parametric surfaces can be simply defined by mathematical formula such as quadric surfaces or convolution surfaces. But the ones used most often are interpolation and approximation surfaces such as Bézier surfaces, B-splines or NURBS. These functions are defined to pass as close as possible a set of control points. An object is then characterized by a mesh of control points and a set of parameters that control the surface behavior.

However, whatever the technique used to define an object, for displaying it, the computer has to sample it into triangle meshes.

Viewing objects

This part will briefly describe the different operations computers have to deal with in order to transform the 3D representation of an object into the 2D image displayed on a computer screen. First, the position of the virtual eye as to be chosen in order to define the origin of the object space. Then, transformations such as rotation, translation, and scale can be applied to the object in order to place it in the desired position. To obtain the 2D image, the size and the distance from the virtual eye of the visualization window has to be determined. An appropriate projection can then finally be applied to the object. All these operations are done in the homogeneous coordinates of three-dimensional projective geometry by simple products of 4x4 matrix. Figure 4.1 describes these different steps.

Rendering objects

To draw a green sphere, you can not just color each pixel of the sphere in green, otherwise you will just obtain a green disk. What is important to render the volume of an object is the color intensity repartition on its surface. This part will then describe the most currently used illumination model in animation application. In this illumination model, the color of an object is characterized by three kinds of reflection:

Ambient reflection The ambient reflection corresponds to non-directional source of light, e. g. the product of multiple reflections of light from the many surfaces present in the environment. Although objects illuminated by ambient light are more or less brightly lit in direct proportion to the ambient intensity, they are still uniformly illuminated across their surfaces. The amount of ambient light reflected from an objects surface is determined by a factor (Ka), namely the ambient-reflection coefficient. This constant, between 0 and 1, varies from one material to another.

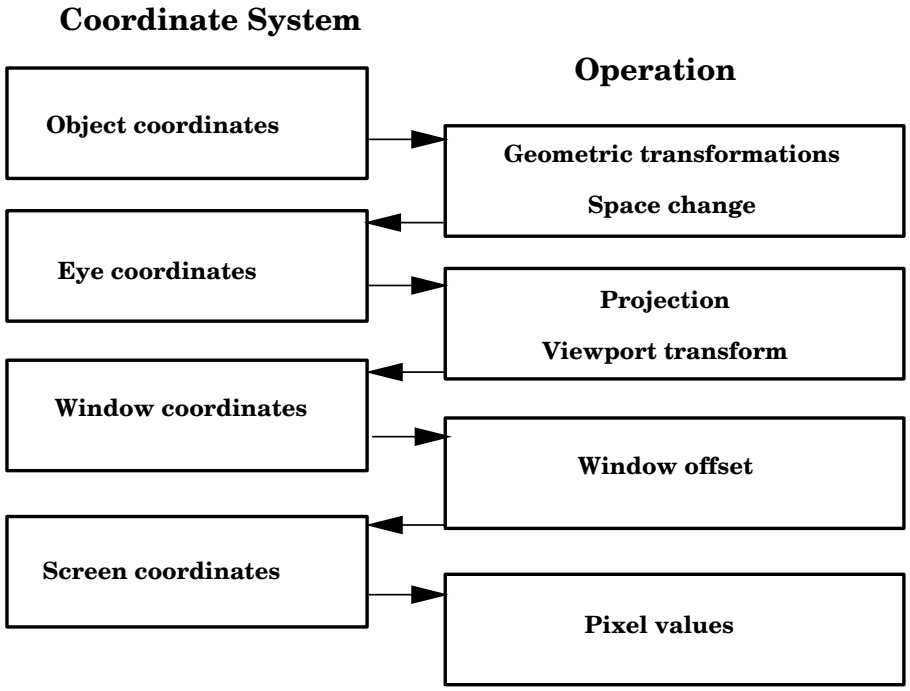


Figure 4.1: Coordinate systems and operations used at different stages of the drawing process

Diffuse reflection Now consider illuminating an object by a point light source, whose rays emanate uniformly in all direction from a single point. The object's brightness varies from one part to another, depending on the direction of and distance to the light source. This phenomenon is known as diffuse reflection. For a given surface, the brightness of a vertex depends only on the angle (i) between the direction (L) to the light source and the surface normal (N) at this point (see figure 4.2). The amount of diffuse light reflected from an object's surface is determined by a factor (Kd), namely the diffuse-reflection coefficient. This constant, between 0 and 1, varies from one material to another.

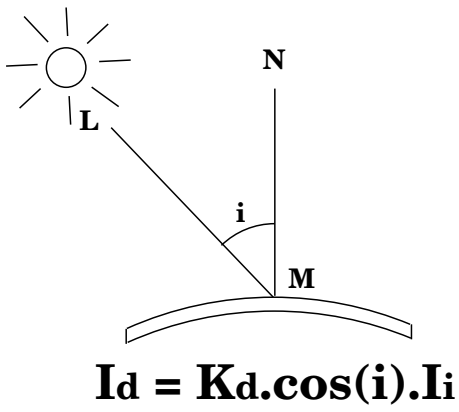


Figure 4.2: Diffuse reflection

Specular reflection Specular reflection can be observed on any shiny surface. Illuminate an apple with a bright white light: The highlight is caused by specular reflection, whereas the light reflected from the rest of the apple is the result of diffuse reflection. The specular reflection represents the fact that shiny surfaces reflect light unequally in different directions. On a perfectly shiny surface, such as a perfect mirror, light is reflected only in the direction (R) which is the mirrored direction about the normal (N) of the direction (L) to the light source. Defining (s) as the angle between (R) and the direction of the viewpoint (V) a popular illumination model for non-perfect reflectors assumes that maximum specular reflectance occurs when the angle (s) is zero and falls off sharply as (s) increases (see figure 4.3). This rapid falloff can be approximated by $cos^n(s)$, where n is the material specular exponent. The amount of specular light reflected from an objects surface is also determined by a factor (Ks), between 0 and 1, namely the specular-reflection coefficient.

It should be clear that we can shade any surface by calculating the surface normal at each visible point and applying the illumination model at that point. This brute-force shading

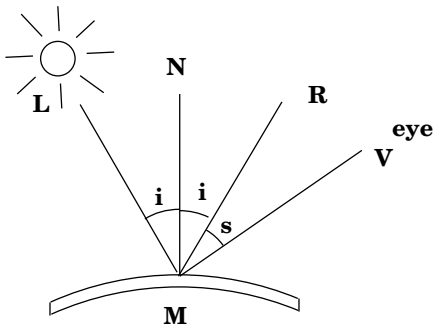


Figure 4.3: Specular reflection

model is expensive except for polygon meshes. As a matter of fact, all the vertices of a same polygon share the same normal. However, in this case all vertices will also have the same color, resulting in a flat shading that does not produce the variations in shade across the polygon that should occur in normal situations.

To prevent this effect, a technique named *Gouraud Shading* can be used. The Gouraud shading process requires that the normal be known for each vertex of the polygonal mesh. If the vertex normals are not stored with the mesh and cannot be determined directly from the actual surface, then we can approximate them by averaging the surface normals of all polygonal facets sharing each vertex. The next step in Gouraud shading is to find vertex intensities by using the vertex normals with the illumination model. Finally, each polygon is shaded by linear interpolation of vertex intensities along each edge and then between edges along each scan line (see figure 4.4).

Another widely used technique for rendering an object is called *texture mapping*. This technique consists of gluing a 2D image onto a 3D object. Actually, a texture can be applied to a surface in different ways. It can be painted on directly (like a decal placed on the surface), it can be used to modulate the color the surface would have been painted otherwise, or it can be used to blend the texture color with the surface color. Texture mapping is often used to represent large scene with a repeated motif such as a large brick wall or with a complicated look, such as the ground of a flight simulator: in this case vegetation pictures are textured on large polygons. This technique is also used to make polygons appear to be made of natural substances such as skin, wood, or marble.

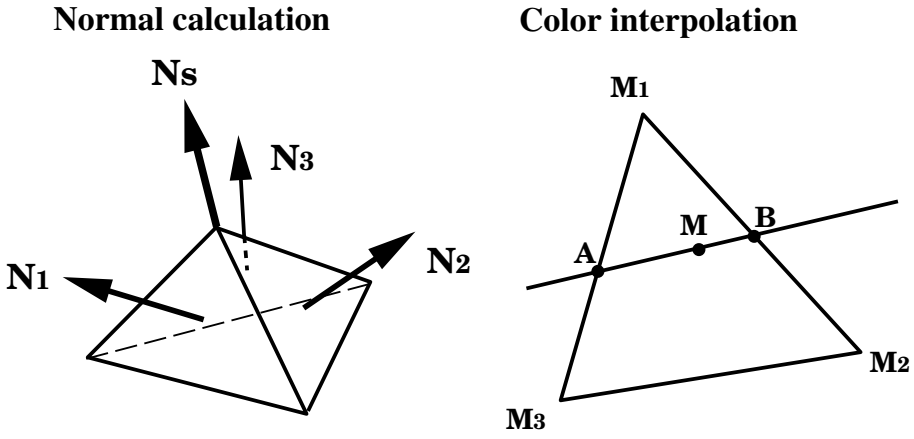


Figure 4.4: Gouraud shading process

4.2.2 The basic Graphics Library OpenGL

As stated earlier (see 1.1.3), OpenGL does not provide high-level commands for describing models of three-dimensional objects. With OpenGL, a desired model must be built up from a small set of geometric primitives [18]. The following parts describe briefly how an 3D object can been drawn.

Defining and storing objects in OpenGL

OpenGL has routines to draw an object in two different manners: by polygon meshes or by approximation surfaces. In the first case, the vertices of the object have to be stored in a file as well as the way they have to be connected together in triangles or four-sided polygons. The normals of the surface at each vertex can also be stored in a file or can be directly calculated within the program. OpenGL provides low-level routines to draw triangles or four-sided polygons meshes by connecting all vertices subsequently in the correct order. During this procedure, the corresponding normals have to be indicated too.

In the second case, the control points mesh as to be stored as well as the control parameters. OpenGL provides then routines to draw Bézier surfaces or NURBS.

Viewing objects in OpenGL

In order to view the object, the programmer has to characterize precisely all the transformations described previously: the place and direction of the virtual eye, the transformations applied to the object, the type of projection, the size and place of the viewport.

Low-level routines are used, and a special attention has to be paid in order to be sure that the object is in the vision field of the eye, otherwise nothing will be drawn.

Rendering objects in OpenGL

OpenGL provides two techniques in order to render an object: Gouraud shading with a classical illumination model and texture mapping. The shading method requires first lights to be defined, placed, and characterized (type, color, direction). Then, the different coefficients described previously (ambient reflection, diffuse reflection, specular reflection, specular exponent) have to be defined. Other parameters can also characterize an object such as transparency or light-emission (for an object such as a bulb, for example). The computer has now necessary information to render the scene.

As texture mapping is concerned, an image as to be provided and the way it has to be textured onto the object has to be indicated.

4.2.3 The OpenInventor graphics toolkit

Open Inventor is an object-oriented toolkit based on OpenGL that provides objects and methods for creating interactive three-dimensional graphics applications [29, 30, 31]. As previously said (see 1.1.4), building a 3D scene with OpenInventor consists of creating a scene database organized like a tree. All information about the scene (object shape, size, coloring, surface texture, location in 3D space, camera place) is stored as nodes of the tree. Then, for the programmer's point of view, all the different steps described previously to render a 3D object are treated on the same level. The next sections will describe the basic nodes used to display 3D objects. In addition, OpenInventor supports the creation of new types of nodes, e. g. describing acoustical or haptic properties of objects.

Defining and storing objects in OpenInventor

As for OpenGL, an object can be drawn with a polygonal mesh or with an approximation surface such as NURBS. Predefined geometric objects such as sphere, cube, cylinder, or cone can also be used in order to construct complex objects. To build a polygonal mesh object, several nodes have to be gathered, each one corresponding to a single property: one node for the vertex coordinates, one node for the normal coordinates, and one node for the type (triangles or four-sided polygons) of mesh and for the connection order. Same principles are used for NURBS representation.

Viewing objects in OpenInventor

To view an object, OpenInventor provides camera nodes that embeds all information necessary to visualize the scene. Then, a type of camera has to be chosen and all the parameters are established through internal methods. This technique is a very useful way for viewing an object and allows very easy camera movements.

Rendering objects in OpenInventor

The rendering part is also quite easy with OpenInventor. A specific light is characterized by a node which contains all information about its type, place, and color. A material node is used for object specification defining the different reflection parameters. Nodes for texture mapping are also provided to enable a very easy use of this rendering technique.

The OpenInventor file format

OpenInventor provides a very useful way to write and to store a complete scene graph in a file, and to read this file later for using it within a program. An Inventor file format has been defined in order to exchange complex scene database between processes very easily. Examples of such file will be described in following sections.

4.3 Representation of Acoustical Properties

Some of the objects used in MIAMI have acoustical properties. The bimodal Text-to-Speech system generates raw audio data and parameters for the face animation. In the analogical demonstrator collisions of the robot with a wall might be sonified by a noise of collision of two materials or the distance of the robot to the wall or the target might be sonified using a sound with a pitch or timbre change according to the distance. In the symbolical demonstrator, locations in the information-city might be identified by an short musical motive when the user is orienting to that place. If somebody want to talk to the user via the network a sound of "knocking at your door" might be played. These examples only give a very brief overview of the different acoustical properties each object has.

The acoustical objects can be classified in the following way:

- continuous streams (e. g. speech)
- non-parametrized single event sounds (e. g. collisions)

- parametrized sounds (e. g. single tone, which changes pitch)
- auditory icons (e. g. short motives)

They will be described in more detail in the following paragraphs.

Continuous streams are represented by the frames of audio samples, which are sent. The sending process sends the data to the audio application server. The exchange can either be done using PVM-messages or reading/writing from the same file. For sending data from one task to another both tasks have to know the TID of each other.

Non-parametrized single event sounds like collision noise are stored as raw audio data in the memory of the computer, the hard disk (as single file), or as specific MIDI sound in a sampler. Depending on the method used, the sounds can be addressed by index-numbers (memory), filenames (hard disk) or MIDI-numbers (MIDI-devices). In the MIAMI project a sound from a database with collision sounds can be identified by the filename.

Parametrized sounds can be sampled sounds, which are played back at different sampling rates using special hardware or realized as software or hardware sound generators. The parameter for the latter case are depending on the specific algorithm used for sound synthesis. The MIDI-protocol is a standardized method for representing the acoustical properties of sounds (timbre and event timing). The most important parameter is pitch. Also the modulation frequency of the amplitude might be useful. In the MIAMI-project simple waveform generators (sine, rectangle, sawtooth, noise) are implemented. For all waveforms the frequency can be changed. For sawtooth and rectangle also the exact shape can be defined (e. g. duty cycle).

Auditory icons can be stored in two ways. A MIDI-sequence stored in a file is sent to a MIDI-device, when the auditory icon has to played. Alternatively the auditory icons can be pre-recorded, stored in a file and the raw data can be played back via the DA-converter. The audio application server supports the playback of files. Each icon is represented by its filename.

In addition to the properties mentioned above the location of a sound objects can be controlled. A direction is usually described by the azimuth and elevation relative to the listener. The sound of each acoustical object which is identified by an integer number can be filtered in a way that it is perceived in the desired direction. The MIAMI-audio

application server supports simple spatialization of sounds either sent from outside the server or generated internally.

For all types of sound the loudness of the perceived sound can be set by setting the amplification of each audio stream. The amplification is represented by a float number.

4.4 Managing Haptic Features

> "Whereas synthetic visual and audio images are omnipresent nowadays, opportunities to reach out and feel non-existent objects possessing texture, shape and inertia are, to put it mildly, not." [2]

Haptic features are not widely used in current applications, although they play an important role in the 'real world'. Three major reasons for this situation have been identified:

i) haptic devices are either not very powerful or very expensive;

ii) sophisticated methods for the generation of haptic feedback are missing;

iii) haptic features are not needed/useful for most computer applications.

In the following paragraphs, we will take a closer look at these arguments.

Devices

The first argument is supported by the observation that most devices with tactile or force feedback fall in one of three categories: 1) not available (prototypes only); 2) inadequate performance; or 3) too expensive. Nearly all devices with haptic output have been either developed for graphical or robotic applications. Many different design principles have been investigated, but the optimal solution has not been found yet.

A device which might help to overcome this situation is the PHANToM, probably the most promising development in this area in the last years [14]. The PHANToM is a 3D input device which can be operated by the finger tip. Its price is US$ 19,500, which is rather cheap compared to other devices with similar functionality. It realizes three degrees of freedom (translational axis) only, but it has many advantages compared to other devices, like low friction, low mass, and minimized unbalanced weight. Therefore, even stiffness and textures can be experienced. Hopefully, the increasing number and popularity of Virtual Reality systems will push the development of haptic devices to a new dimension.

Methods

The lack of good methods is related to the lack of good (and widely available) devices. Usually, for the large number of prototypical devices existing in many laboratories, a large number of prototypical methods has been developed, whereas general methods are still missing. Even worse, many devices and methods have been developed for special applications only, thus they are not useful in other contexts.

Again, the team around Salisbury might show a way out of the dilemma [23]. They have investigated principle methods for so-called *haptic rendering*, thereby using and adapting well-known rendering methods from computer graphics. The first approach has considered free space movement, contact situations, and methods for modeling an object's surface. Although developed for and tested with the PHANToM device, this approach might be useful for other devices as well:

> "While researchers have begun to look at algorithms for generating forces resulting from contact with virtual objects ..., we feel that there is a great need for a more coherent approach to generating ... these sensations and modeling interactions with complex objects. Our interest is in developing a framework in which we may represent shape, surface properties, bulk properties and multiple object interactions. ..." [23, p. 124]

Applications

Whether haptic feedback is needed or useful in an application depends to some extend on the application and the available device(s). Until now, it is mainly used in special applications with interactions in 3D, like telemanipulation, CAD, or virtual reality. These applications usually need a powerful graphics workstation and special hard- and software, therefore the price for an input device with haptic feedback does not play a central role.

This is completely different for everyday applications. Even if good devices and methods would exist, the remaining question would be: *"How much money would an average user spend for this device and method?"*

Therefore, in the MIAMI project we decided to follow two different approaches: First, the FORCEJOYSTICK is used for controlling a mobile robot (**??**). In the second application, the FORCEMOUSE is used to support user interactions in any Tcl/Tk application by providing haptic feedback (**??**). Both devices are relatively cheap and have been developed within the first year of MIAMI. The different requirements of both scenarios will be described in more detail in the following paragraphs.

4.4.1 Robot navigation with force feedback

The main intention here is to support the operator during interactive control of the platform in order to avoid collisions and to simplify the navigation in narrow passages like doorways. First tests have shown that satisfying results can be achieved only when several methods will be combined. Therefore, we started to develop a system which calculates a force feedback vector based on the following inputs:

- a global algorithm calculates boundaries for all the objects in the robot's environment, based on the robot's size and orientation;
- local planners consider the robot's direction, orientation, velocity, and input of the operator;
- different rules based on heuristics are applied in different situations, e.g. when the robot passes a doorway.

In contrast to several developments directed towards autonomous movement and navigation of mobile platforms, our approach has been especially designed for interactive operations. The focus has been set to the development of an overall concept for robot navigation with force feedback realized by the methods described above. The FORCE-JOYSTICK is used to prove the principle idea, although it is by no means a sophisticated device. The main goal is to support the operator as much as possible, therefore we plan to integrate other kinds of feedback in later versions of this application, too.

4.4.2 Intelligent support of 2D interactions with haptic feedback

Usually, a mouse is used for input activities only, whereas output from the computer is sent via the monitor and one or two loudspeakers. But why not use the mouse for output, too? For instance, if it would be possible to predict the next interaction object the user wants to click on, a mouse with a mechanical brake could stop the cursor movement at the desired position. This kind of aid is especially attractive for small targets like resize handles of windows or small buttons.

The specialized FORCEMOUSE developed within the project's first year has simple (and cheap) extensions which provide haptic feedback. With this device, we have carried out some basic experiments. The results revealed that haptic feedback is useful only if it is supported by an intelligent control mechanism, at least if applied to complex systems like GUIs. Therefore, we have developed a multi-agent system which analyzes the user's

interactions for some time, generates a user- and application specific model, and uses this model to predict the next interaction. Finally, this knowledge is used to support the current positioning task by stopping the user's movement. Although primarily designed in order to provide 'intelligent' *haptic* feedback, the system can be combined with other output modalities as well, due to its modular and flexible architecture. The implementation is a plug-and-play solution for any application written in Tcl/Tk [17].

The results of the experiments performed in WP2 helped us to identify the drawbacks of a simple and the requirements of a good controller for the FORCEMOUSE:

Reaction time: In order to present the user a consistent and coherent feedback, the delay between entering a widget and launching the feedback should be minimized.

Prediction: The best result regarding the reaction time can be achieved when the next widget is known in advance (*a priori*). Therefore, a prediction mechanism is needed.

Adaptability: Every application uses different widgets with respect to size, position, etc., and every user has a different way of interacting with an application. Therefore, the system should not only model the user's behavior with respect to a specific GUI but should also adapt itself, thus increasing its performance and decreasing the error rate over time.

Independence: Although the statistical models are based on a specific user's actions in a specific application, the system itself should be usable in combination with any Tcl/Tk application without any changes.

Versatility: The system architecture should be modular and flexible in order to support the integration of other output modalities, too. In addition, the statistical model might be used for other purposes as well, for instance for recording macro operations or generalizing action patterns.

The principle idea of our approach is to predict the next user action regarding the usage of widgets in the graphical user interface, i. e. to predict which widget will be used next. In other words:

> *The main task is to* **predict** *the next user action in order to launch the haptic feedback* **selectively** *and to* **adapt** *this capability over time.*

4.5 Multimodal Integration

Multimodal integration can be based on an amodal, a transmodal, or a paramodal representation. *Amodal representation* involves objects defined in terms of internal (physical) properties which have simultaneous influence on the two (or more) modalities in which the object is represented, e.g., a vibrating cord, or a model of the vocal tract. *Transmodal representation* applies to objects which intrinsic properties can be converted from one modality to the other, e.g., moving lips animated from the acoustic speech signal. *Paramodal representation* suggests that the object has a set of unimodal intrinsic properties independent from one another, except for the time relation that control them, e.g., a jack-pot and its noise.

Most of the multimodal systems belong to the third category. In this case, the main challenge is to ensure a perfect synchronicity between the modalities so that changes occur at the same time, whatever the sampling frequency used in each modality may be. In the following sections, five applications which have been designed to demonstrate the multimodal integration and the combination of modalities and software packages will be presented. In addition, you will find descriptions of the basic objects defined for these applications.

Chapter 5

Navigation

5.1 Navigation in Hyperspace and Cognitive Representation

The theme of navigation in information space was identified in an early stage of the MIAMI project proposal as an essential point of study. Navigation can be described as a process of movement and orientation, yielding a trajectory that is directed towards a given goal. However, navigation also appears to be a highly ambiguous concept. What is the actual space in which is navigated? Is this a graphically isomorphous and three-dimensional environment? Does the space of the navigation process consist of a two-dimensional graphical screen contents viewed in a sequence? Or should we, alternatively, view the navigation in computer use as a process of cognitive movement through a more abstract information space, pondering on some topics and wandering over others? Thus, there is a distinction between physical and conceptual navigation. The question within MIAMI is what happens if the navigation is both physical and more abstract or conceptual at the same time? As an example, as user may have a fuzzy idea (the conceptual level) about the type of video movie which would be interesting to view this evening, and is confronted with a graphical user interface (the physical level) designed to navigate to the actual best-fitting available material, given the viewer's goal. To what extent is it helpful to use graphical artefacts to improve the process of navigation under such circumstances?

In this section we will deal with the subject on how people build up mental representations of information which they during a hypermedia tour. Thus, we direct our attention to navigation in an essentially non-geographical environment. The general theme (as addressed in WT 3.5) is 'navigation in information space', but we will focus our attention to the typical use of World-Wide Web browsers. A typical problem is the fact that users are

'lost in hyperspace' after only a few 'clicks' on hyperlinks. The human short-term memory is known to hold about 7±2 items [16, 4]. After a few pages, new information is processed at the expense of the information collected earlier. What is even more, is the fact that the current **User Goal** also requires cognitive space or resources in the short-term memory (STM). It is very likely that the user loses him/herself in futile browsing, forgetting about the actual original goal of the information search action.

Typically, humans have tried to solve the problem of limited STM by creating cognitive artefacts which are persistent and visual, like sketched maps and the graphical symbols of the alphabet. So, maybe also in 'computer navigation', we may want to compensate the low capacity of the STM by using visualisation techniques which make the structure of a hypermedia document more explicit and which make navigation easier.

One solution to the problem of the visualisation of document structure is introduced by Shneiderman [26]. He proposes a number of tree visualisation techniques (Tree-Maps). By making graphical objects in such a tree 'clickable', the static tree image representation turns into a dynamic tool. As Balasubramanian [1] points out,

> "Graphical browsers help reduce disorientation by providing a two-dimensional spatial display of the hypertext network. They also help minimize cognitive overhead by showing a small part of the network. They also provide an idea about the size of the network which help users estimate the number of nodes and links in the system."

There are basically two approaches in visualisation:

1. The actual provider of the hypermedia document creates the graphical representation of the message structure,
2. An algorithm is used to analyse an ordered list of links, i.e., the recent navigation history for this user, followed by subsequent automated visualisation.

The problem with the second approach is that it is unclear whether it is at all possible to infer a meaningful, conceptual (semantical) structure from a temporal link-access order. Examples of superficial automatic visualisation of the hypertext navigation trail are the WebMap program by Peter Dömel [6]. Another experiment is based on a Perl program developed by Paul Harrington, using the automatic 2-D layout system for undirected graphs *Neato* by Stephen North from AT&T.

The SGI File System Navigator (fsn) demo program (known from the Jurassic Park movie), is another visualisation solution where the hierarchical directory tree structure

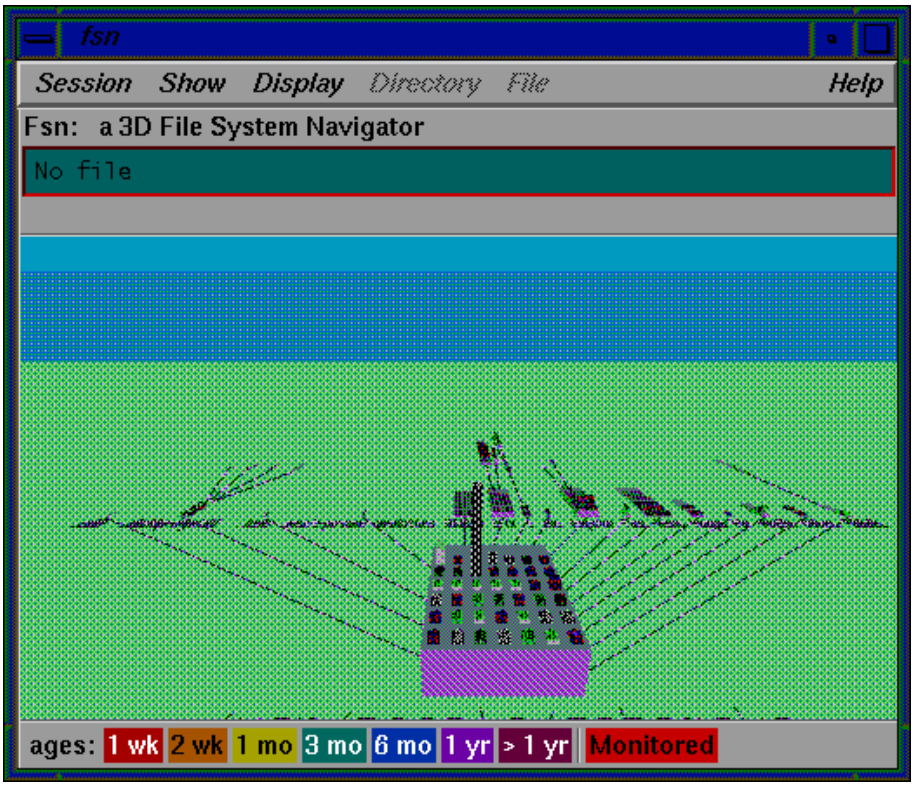


Figure 5.1: A screen dump of the three-dimensional file manager program *fsn* from Silicon Graphics. The hierarchical structure of the graph is projected in 3D, allowing the user to 'fly over' the file system.

of a file system is visualized in 3D (Figure 5.1). Whether this actually helps the user to build up a good mental representation remains to be seen, but fsn certainly gives an interesting perspective on your own well-know directory tree. Files are 3D blocks, and the file attributes are displayed as color size, and icons (the latter on the "roof" of a file "building"). However, hypermedia structures are not necessarily hierarchical, which makes things more complicated. Another example of 3D graph visualisation is by Peter Young [5]. In this example, the structure concerns the calling tree of a C program, but theoretically, this 3D approach could be used in visualising WWW neighbourhoods, too.

As Young notes (rephrasing): 'One of the biggest problems when using 3D visualisations in which the user can roam freely is in the viewpoint control and spatial awareness. On the one hand, the user must be able to confidently move from point A to point B in three-dimensional space. On the other hand, they must also be able to determine their absolute position or orientation if they become lost'. As an example, it is very easy in a 3D graph to overshoot some nodes, suddenly being faced with a blank wall or panorama,

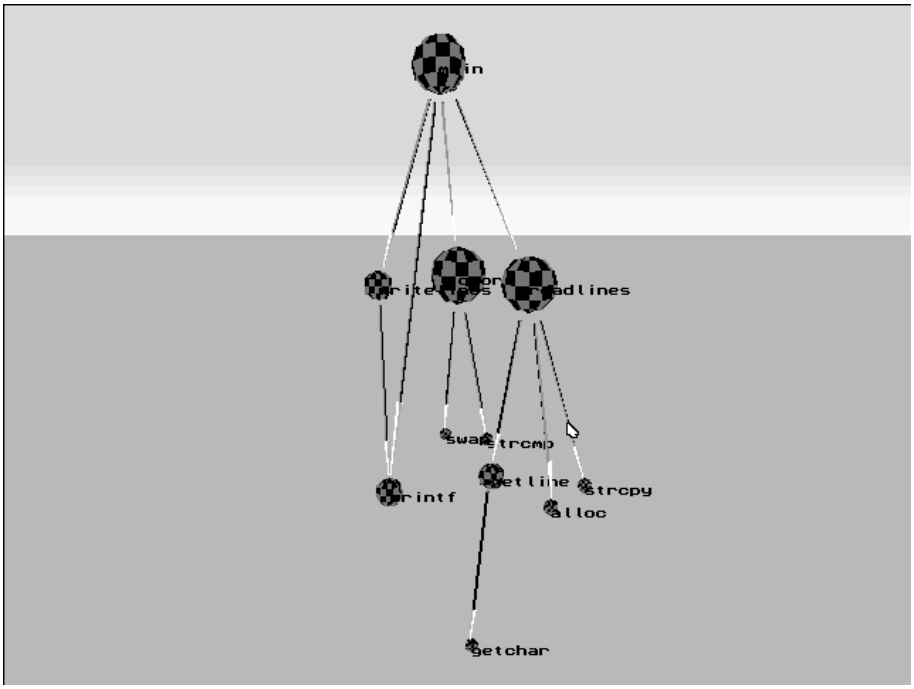


Figure 5.2: A three-dimensional graph of a program in C (courtesy Peter Young, Univ. of Durham/Centre for Software Maintenance, UK).

after which one must turn 180 degrees to see the structure again, now from a new angle. The conclusion of this is that if we are already confronted with an *information-space navigation* problem, the adding of *three-dimensional navigation* may actually deteriorate the navigation process.

A second category of graph visualisation is the Venn-diagram. Such a visualisation is only possible if there is a strict hierarchical structure (Figure 5.3).

The TreeViz idea of Shneiderman belongs to this category. Figure 5.4 gives an example of TreeViz-type layout, which is effectively a rectangular approach to the typical oval-shaped Venn diagrams. As can be seen clearly, such approaches will not suffice in the case of a network topology, such as is required if a hypermedia navigation trail is to be visualised.

A number of small-scale studies has been performed to explore the effects of visualizing the direct neighbourhood in a hypermedia document by means of a two-dimensional graph.

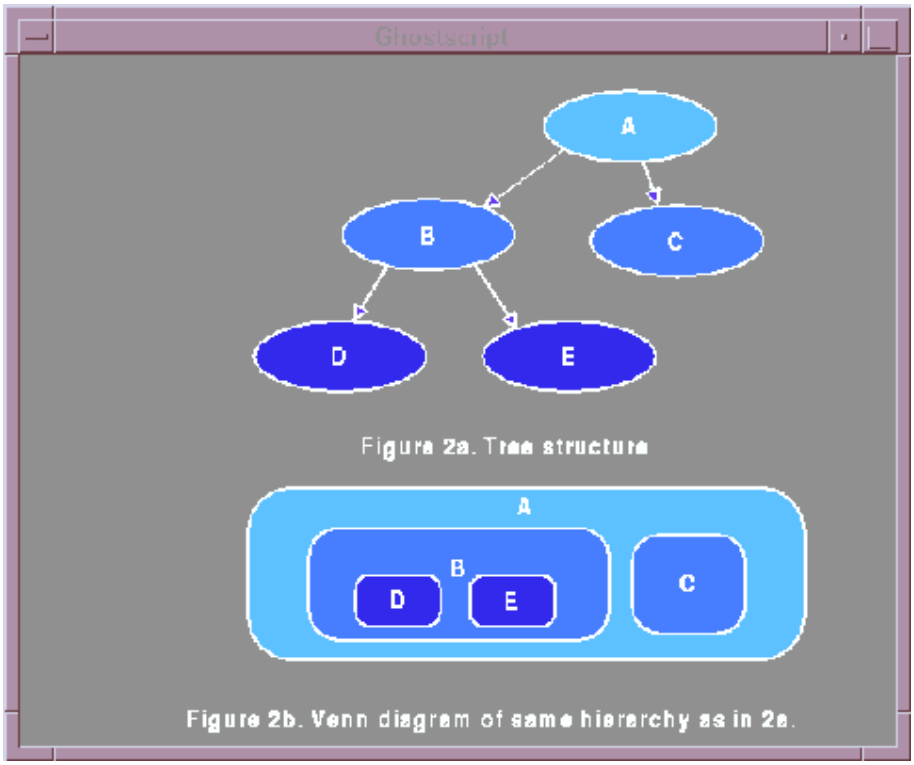


Figure 5.3: A simple tree and its Venn diagram representation.

5.1.1 Presentation of a 2D graph of the neighbourhood topology in hypermedia navigation

Twelve subjects took part in the experiment. Their task was to answer six specific questions on two hypertexts on two well-known bands ("The Beatles" and "The Rolling Stones"). The following is a summary of findings.

- The visual presentation of a graph strongly reduced the number of erroneous page selections (118 irrelevant pages clicked without graph, 67 error selections if a graph is present).
- The total search time is not significantly reduced by the presence of a graph (38s. without graph, 34s. with graph). Apparently the graph interpretation also takes considerable time.
- The nodes in the graph were not made 'clickable'. This was done on purpose, to keep the two conditions more comparable. Many subjects tried to click on the nodes in the graph and did not devote much attention to the graph after finding out that it

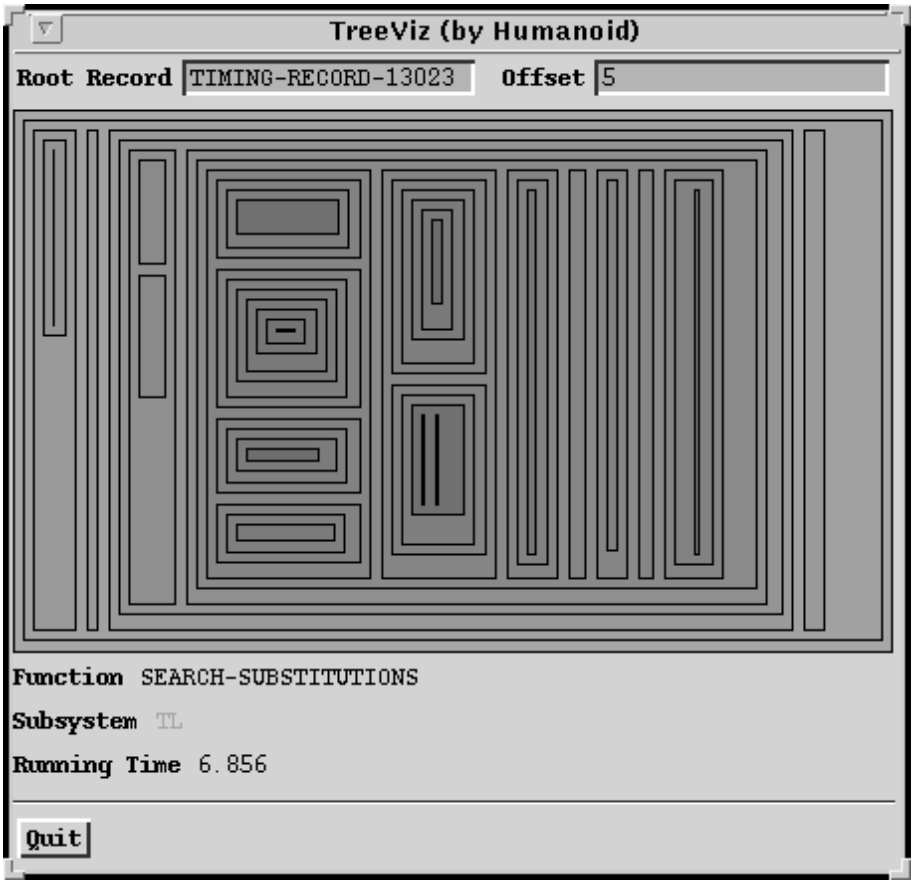


Figure 5.4: A TreeViz representation of a tree structure

was not a navigation tool.

These findings support the idea that a visible graph as a navigation tool may be useful, although the main advantage cannot be expressed in the form of (reduced) navigation time. Replications of this experiment with other content topics and other types of graph visualisation mainly corroborated the findings. However, the graphical quality of the graph seemed important. This is difficult to quantify, but 'visually pleasing graphs' elicited better use than duller versions. A problem with the graphs is that they consume a considerable portion of the screen, which is already mainly occupied by the browser surface itself.

A second study was performed in which the user was forced at regular intervals to maintain a list of current goals in a text-item window next to the browsing program. In this study, a realistic, large hypermedia document on astronomy (our solar system) was used. Also here, the subjects had to answer specific questions concerning the content of the document. At the moment of clicking a hyperlink, the user was asked regularly (not each time) to fill in

a line of text concerning the current search goal. Although this is admittedly an intrusion on the normal flow of interaction, the frequency of this "Goal" dialog was kept low enough to prevent irritation, and high enough to keep track of the user's activities. The results of this experiment are currently being evaluated.

Chapter 6

Some examples of new 'Building Bricks' in Multimodal Systems

As a more general description of some of the modules presented in chapter 3 we present a number of new 'building bricks' developed in the MIAMI project. They vary from device oriented to environment oriented:

- MDD - the meta-device driver
- GESTE - a two-dimensional gesture recognizer
- AAS - an audio application server
- HARP - a multimodal environment based on cognitive processing

These are just a selection from the much larger list of components which are currently under development.

6.1 *MDD*—unified access to different devices

The Meta Device Driver (*MDD*) is a C-library which provides unified access to several input devices, some of which also have the capability to generate haptic output. This section describes the concept and structure of the *MDD*. A detailed description of all functions provided by the library and an explanation how to use the functions in an application and how to extend the *MDD* in order to include other devices is available on-line on the WWW.

6.1.1 Idea and Concept

Each device driver for a specific I/O device provides different functions and features, and the functions use different sets of parameters. Therefore, for every new I/O device functions to access it have to be included in *every application* using the device. The basic idea of the META DEVICE DRIVER is to provide unified access via a general interface to various devices. Nevertheless, most features of the devices should still be accessible, which means to find a balance between generality and specificity.

Specific device drivers A device driver connects an I/O device to an application and provides a channel to exchange information in one or both directions. Therefore, a communication protocol with a specific coding of commands and parameters will be established. Usually, the application sends a command or request and waits for an acknowledgment or some values (see figure 6.1), but some I/O devices also send their values permanently.

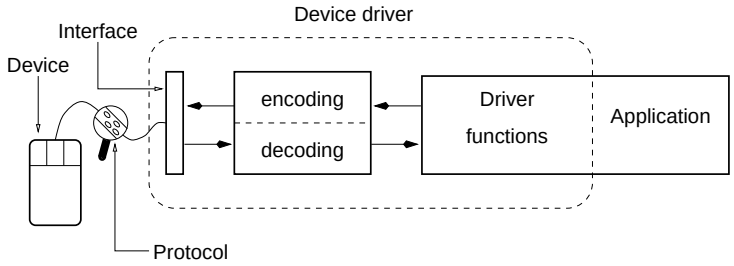


Figure 6.1: Principal structure of a device driver

The obvious advantage of this kind of driver is its specificity: the driver provides access to one specific device only. Thus, exactly all features of the device are supported and the performance can be maximized because the communication overhead can be minimized—the driver is the only layer between the device and the application.

The META DEVICE DRIVER The META DEVICE DRIVER's job is to provide access to more than one device via a general interface. It introduces an additional layer between the specific device drivers (SDDs) and the application. On the one hand, the *MDD* exploits the SDDs' functions for communicating with a specific device. On the other hand, it supports a small number of general access functions to be used by the application. Therefore, the data transmitted between the devices and the application has to be converted in a general format. The principle structure of the *MDD* is shown in figure 6.2.

The advantage of unified access to several different I/O devices introduces two problems:

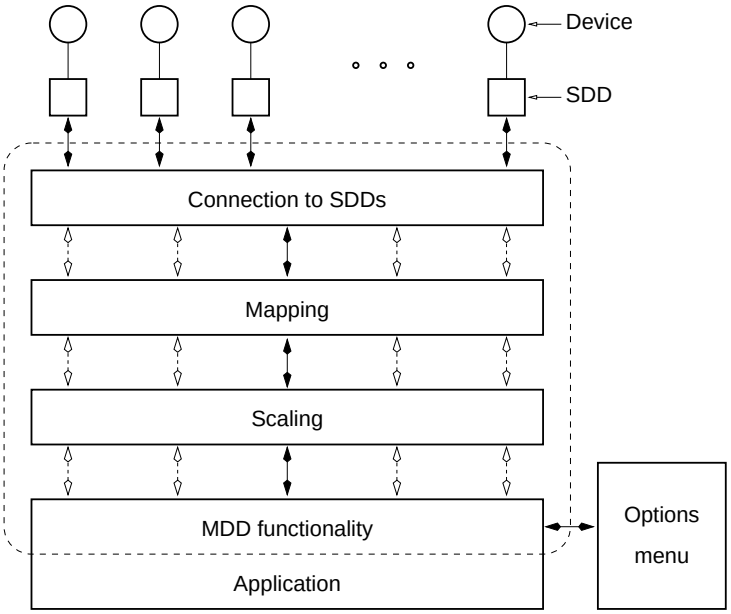


Figure 6.2: Principal structure of the META DEVICE DRIVER

First, some of the specific features of a device might get lost due to the general functions provided by the *MDD*. This depends strongly on the respective devices and their drivers. Second, the additional layer might introduce an additional time delay in the communication process. Although this is true, the delay is no serious drawback and is for most applications fully compensated by the advantage of the general, simple, and consistent interface[1].

Architecture of the *MDD* In order to design a general interface, the specific characteristics of several I/O devices have been analyzed. Regarding the input space, a superset is easy to construct using an input space with six dimensions: three translational and three rotational axes are needed to reach any position in 3D space with any orientation. To simplify the access, these values can either be provided in absolute or relative format. If a device does not provide six degrees of freedom, the missing axis can be disabled by a masking process.

The generalization of the output space is significantly more difficult, because the output capabilities of the devices vary much more. Although in this case only input devices with output capabilities are considered (i. e., no monitors, no loudspeakers, etc.), the haptic output is very specific to each device. Therefore, three different functions have

[1] In fact, the delay is usually only a few milliseconds, whereas the differences in the performance of the various devices is much larger, ranging from < 10 to ~ 100 ms!

been implemented which allow to provide force vectors in 3D space, "gravitation vectors" in 3D space, and vibrations, resp.

Due to the ability of a META DEVICE DRIVER to communicate with several devices, so-called *logic ports (LPs)* have been introduced[2]. Thus, an application can open multiple channels in parallel and communicate with more than one device. The motivation was to provide a way to change the devices "on the fly" and to select the best device at any time of the interaction process. To emphasize this feature, a *composition* function has been realized which combines two LPs into one. In this case, the application receives values from two devices without noticing a difference.

It is also possible to use the *mapping* function in order to map one axis to another or to create a 4D device from two 2D devices on a logical level. The main purpose of mapping is to map the specific axes of a device to the internal axes of an application, e. g. if the displayed manipulations on a monitor are not consistent with the manipulations of the device.

After mapping, the values will be processed by a *scaling* function. Thus, the input values of a device can be adapted to the application and the user's needs, e. g. by increasing or decreasing the "speed" of one or more axes. By using negative scaling values, the direction of an axis will be inverted. Both, the mapping and scaling function, can be called by the application or via a graphical user interface, the so-called *options menu*. It will be described in the next section.

All features of the *MDD* can be used at once simply by linking the *MDD* library to an application.

6.1.2 Implementation

The META DEVICE DRIVER has been implemented in C, using PVM (see 1.1.5) for inter process communication and TkPVM (see 1.2.1) to implement the options menu. The complete structure of an application using the *MDD* library is depicted in figure 6.3. The physical driver which realizes the lowest level of communication are separated processes which communicate with their specific modules via PVM. The modules are linked to the *MDD*, which again is linked to the application, thus providing easy access to several devices via general functions.

In order to support completely independent modules for the specific device drivers (SDDs), function calls from the *MDD* to the SDDs are mapped via a structure which uses pointers to functions. This concept allows the easy integration of other SDDs by following a simple

[2] An LP is very similar to a file handle.

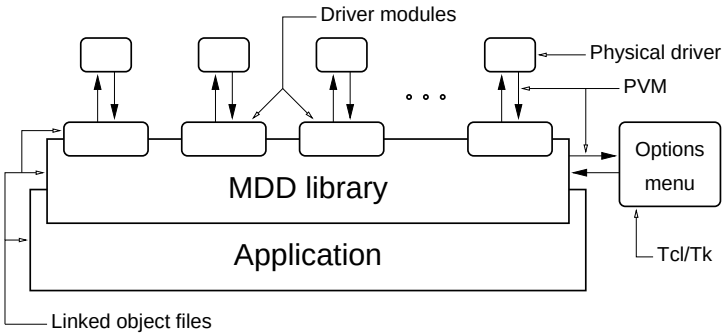


Figure 6.3: Structure of an application using the *MDD*

scheme.

Procedures and commands In the following, the *MDD*'s interface will be described briefly.

mdd-connect Establishes a connection to a device. The device is connected to physical port *portName* on host *hostName*, and a logic port number to be used by the application in further steps is returned.

If the device is ID_COMPOSE, the following two parameters do not specify a port and a host but two logic port numbers which are composed into one.

mdd_disconnect Cancels the connection.

mdd_get Absolute input data is received in *data*, considering the degrees of freedom specified in *mask* only. The button state is always returned for all buttons.

mdd_getDelta Like mdd_get, but returns relative values in *data*.

mdd_setForceVector Creates a force vector with strength *strength*. *strength* must be a value in the range of 0 to 100, which means 0–100% of the force that the device can apply. The direction of the force vector is specified as translations and rotations in *data*. If necessary, particular degrees of freedom can be ignored by specifying a *mask*.

mdd_setForcePoint Applies a gravitational force with strength *strength*. *strength* must be a value in the range of 0 to 100, which means 0–100% of the force that the device can apply. The center of gravity is specified as translations and rotations in *data*. If necessary, particular degrees of freedom can be ignored by specifying a *mask*.

mdd_setVibraton Starts a vibration with a period of *period* milliseconds and a duration of *duration* semi-oscillations. Thus, single impulses can be applied, too. If *duration* is -1, the vibration is switched on permanently and must be switched off explicitly by the application.

mdd_setForceOff Switches off all haptic output.

mdd_setOption Changes the options settings. Several options can be changed with one call, because the number of arguments to mdd_setOption is not fixed. It is important to notice that the mapping is applied before the scaling, that is scaling is applied to logical axes.

mdd_Options Opens the options menu (see below).

mdd_OptionsApply Applies the current values of the options menu to the *MDD* functions.

mdd_OptionsEnd Closes the options menu. Attention: The values are **not** applied by calling this function!

6.1.3 The options menu

The options menu provides a graphical user interface to the *MDD*'s mapping and scaling functions. It has been implemented using TkPVM (1.2.1), which means that the graphics part is realized with Tcl/Tk (1.1.2), and it is connected to the *MDD* via socket communication using PVM (1.1.5).

Because the options menu is a mere *slave process*, it can not apply its current settings and it can not be closed by the user at the interface level. Instead, the values have to be fetched by the application which is also responsible for closing the menu. A typical interaction for changing some settings looks like this (compare to figure 6.4):

1. Open the options menu by calling `mdd_Options`. The current settings will be used to initialize the display. The logic port number is displayed in the title of the top-level window.
2. Adjust the values and the mapping as desired.
3. Call `mdd_OptionsApply` in order to apply the values to your application. Remember that these settings are valid for the specified logic port only.
4. Repeat steps 2 and 3 if necessary.

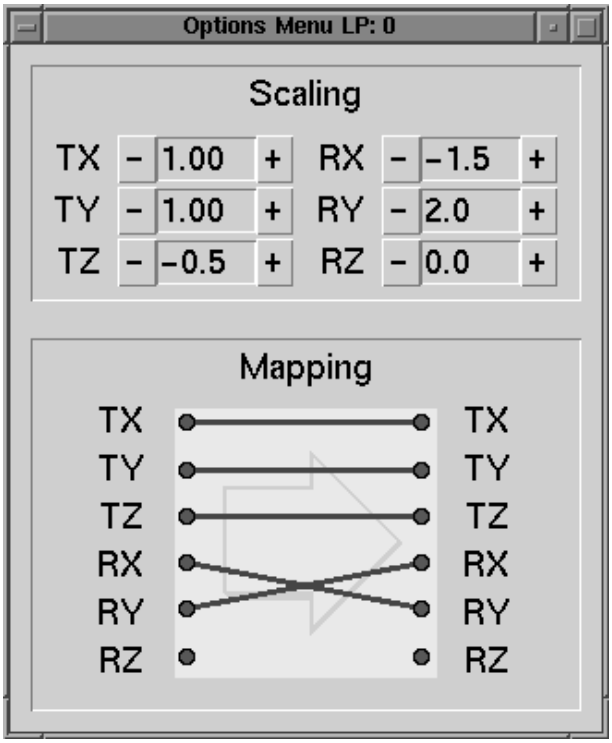


Figure 6.4: The options menu

5. To close the menu, call `mdd_OptionsEnd`. Remember that the values are applied only if you call `mdd_OptionsApply` *before* closing the menu!

6.2 *GESTE*, a simple classifier for two-dimensional gestures

This package or building brick performs a demonstration of basic functions in pen-based interfacing, which can be easily extended to a multimodal domain. Its function is centered around classification of the shape of time-bounded trajectories in a plane, such as handwriting gestures or 2D gestures of non-handwriting origin. Examples are: Mouse, joystick, finger-tip movement on a touch-sensitive surface or the 2D projection of index finger gestures in front of a camera).

6.2.1 Application examples

This gesture recognizer can be used to recognize user-defined commands, e.g., in a pen-computing setup. Apart from typical editing commands, such as gestures for the *Insert*, *Cut*, and *Paste* commands, which can be found in pen-based operating systems such as Windows for Pen and PenPoint, we have used gestures for executing a function like the *Give-Other-Words* command to a cursive recognizer: A user writes a cursive word which is misrecognized. A simple user-defined gesture may call up a pop-up list of other word alternatives, of which one may be clicked by the pen. This immediately replaces the wrong word with the chosen word, and the writer may continue to write the next word. Another example is the possibility to pop up a phone book by drawing a stylized telephone (*Exec-Phone-Book*), or to dial a number by making a circular spiraling movement (*Dial-Selected-Number*). Furthermore, as reported earlier (Report 2 of MIAMI), this recognizer is suitable for implementing the Goldberg [9] alphabet, with its unistroke alternatives to the alphabet of isolated hand-print characters in Western handwriting.

6.2.2 How *GESTE* works

The gesture recognition method is based on a simple template matching scheme (nearest neighbour) and can be tuned to the user on the basis of the Kohonen LVQ training approach. For simplicity, we will assume the pen as the input device and the origin of the XY-coordinates in the sequel. An essential assumption for the *GESTE* classifier is that the user produces a single curved trajectory. The shape of the trajectory must be sufficiently close to an idealized representation for this class of gestures, and must at the same time be sufficiently far away from the other gesture classes which are in use in the system. Also, the gesture trajectory must be clearly bounded in time, and be separable from the stream of ongoing movement. In fact, we assume a box-car multiplication of a

limited-duration window with the continuous signal of user-induced movement. In the case of the pen, this is usually realised in the form of an axial switch in the stylus. In the case of pen input, the user separates the 'serious' part of the movement from 'non-relevant' movement before and after the gesture by putting the pen on the writing surface, and activating the internal switch. Thus, what actually happens is that the movement is (a) recorded in the X and Y directions, using a digital approximation of a continuous signal, and (b) in the force domain (F), using a binary signal (0/1). The force signal thus delivers the box-car function to select a movement sequence of relevance. In other input devices than the pen, a similar segmentation must take place. If no third dimension is available (only X and Y), simple movement lemmas must be defined to identify the time of start and time of ending of a gesture. A simple convention is the use of a pause in the movement before and after the gesture. Additional certainty concerning the occurrence of a gesture in a continuous stream of movement can be derived from the explicit definition of zones in the user interface surface (or volume) where gestures are allowed. Theoretically, Markov modeling could have been used to monitor a stream of on-going movements and firing a symbol when a reliable sequence is detected. In the current simple approach however we put the segmentation responsibility in the hand of the user. In the case of the pen, as stated before, this segmentation is rather straightforward. Gestures and Blockprint are assumed to be single-pendown traces (sometimes called unistrokes). Thus, writing a capital E with its intermittent pen-up movements is not handled by *GESTE* unless a post processor classifies/parses the stream of recognized basic unistrokes (which are straight lines in the example of the E). A precursor of the *GESTE* recognizer has been used successfully in a demonstration program *PenBlock*.

6.2.3 Assumptions

If movement patterns are well trained, we may assume the presence of what is called a 'motor program' [3]. There are representations in the brain, which catch some invariant properties of the pattern. Notably, points of high curvature in the trajectory are assumed to be important in this respect. The trajectory between two points of high curvature is called a 'velocity-based (VB) stroke[3]'. Several replications of the same movement pattern share enough shape details in common, such that the average time function of several time-normalized replications of the movement will contain the essential shape, provided that the movement pattern does not consist of too many strokes [24]. This principle is known as 'the homothetic assumption' [28]. In *GESTE* we assume a (soft) maximum of six VB strokes. Due to these assumptions, the classification of the gestures works best if

[3]Not to be confounded with the unistroke, which is a pen-down stream of coordinates

the starting and ending points are of low velocity, which is reasonable.

6.2.4 Processing of a 2D gesture

Given an XY trajectory in time, typically sampled with a minimum sampling rate of 100 Hz, the following processing steps are followed.

- The bounding box of the gesture is calculated in the coordinate system of the input device, for later use in the application, such as relating the gesture to objects on the interface surface.
- We heuristically define five sample points per stroke to be sufficient. With five samples per VB stroke, handwriting is still very well legible, whereas this quickly deteriorates for smaller numbers of points. Assuming that the maximum number of VB strokes is about six, we choose a number of points equal to 30, for a whole gesture. The total length λ of the trajectory is calculated and the trajectory is - spatially equidistant - resampled, using $d = \lambda/30$.
- The center of gravity $O = (\mu_x, \mu_y$ is calculated. The standard deviation of the radius r with respect to O of all samples is calculated (σ_r). Then the X- and Y axis are uniformly rescaled such that standard deviation of the radius equals one ($\sigma'_r = 1$). This operation leaves the aspect ratio of the gesture intact and is more stable than normalisation on the basis of the bounding box. We now have for this gesture a 60-dimensional feature vector containing the spatially resampled X and Y with a normalized scale.
- The last step concerns an extension of this gesture feature vector with the running angle of the trajectory in the form of 30-1=29 pairs of $(cos(\phi), sin(\phi))$ values. These can be calculated as $(\Delta x/d, \Delta_y/d)$ without using trigonometric functions, where $\Delta_x = x_k - x_{k-1}$, and d is the sampling increment (see above).

We now have obtained a gesture feature vector with 118 real-valued features:

$$\vec{f} = (x_0, y_0, ..., x_{29}, y_{29}, cos(\phi_1), sin(\phi_1), ..., cos(\phi_{29}), sin(\phi_{29}))$$

Consequently, we may now search the nearest neighbour to $\vec{f}$ in a table $\vec{G}_i$ containing the prototypes of the gesture classes. For this, *GESTE* uses the simple unweighted Euclidean distance. A list of gesture hypotheses is returned, in order of increasing distance, i.e, in order of decreasing likelihood.

6.2.5 Training

The major advantage of the simple approach of *GESTE* is that a user may define his/her own gestures, in an on-line fashion. A new gesture class is defined by adding its feature vector directly as a new entry to the table $\vec{G}_i$. Improving the representation of an existing gesture is done by the Kohonen LVQ scheme, in which the feature vector effectively is a running average:

$$g_{ij}(k) = \eta f_j + (1 - \eta) g_{ij}(k - 1)$$

i.e., the value $g_{ij(k)}$ of feature j of gesture i at training occasion k is a weighted sum of feature j of a new gesture sample and the previous value of that feature, $g_{ij}(k-1)$, on training occasion $k-1$. Here, η is the learning rate, which has a typical value of 0.05.

6.2.6 The current state

Currently *GESTE* is under development. As described in 3, this module receives the XY coordinates via PVM, and sends the recognition results to the application who needs it. For this to succeed, *GESTE* needs the concept of **focus**, i.e., the application must notify *GESTE* which MIAMI PVM module currently has the focus, and needs the results of the gesture classification.

Figure 6.5: A screen dump of the *GESTE* module with a number of experimental gestures.

6.3 AAS - An Audio Application Server for multimedia applications

Complex multimedia applications have special requirements for the auditory modality. Several agents in a multimedia environment might generate sounds or sound events at the same time. All different audio streams have to be added and sent to the hardware specific driver for the audio output. Some operating systems like SUNOS do not allow such applications. If one application has allocated the audio output device, all other applications have to wait until the current client has finished its audio task. This problem does not exist for Silicon graphics workstation. Multiple applications may open individual ports to the driver of the audio device. The signals of the different applications are either added by software or by designated DSP-hardware. The output parameters like sampling rate and gain can only be set for the whole audio output, not for each stream individually.

Current libraries and concepts for audio output also do not make any use of simple stereo mixing techniques or more sophisticated sound spatialization techniques. The human auditory system is able perceive sound sources from any direction around the head. This is very important in daily life, as it allows to discriminate different concurrently active sound sources and to direct the attention to one of the sources located at different positions in space (Cocktail-party effect) (to be used in videoconferencing). The location of a source also can be used to improve the orientation of the user. A simple example in multimedia applications might be the mailbox. If an e-mail arrives, current interfaces will play the sound of a beep over the loudspeaker. The user does not get any information about the content and the position of this application at the two dimensional screen. The noise of a letter falling in a mailbox perceived at the position of the graphical mailbox icon will reduce the time and effort to find the desired information.

Another disadvantage of current audio realizations is, that the audio application has to run on the host, which is generating the audio output. In portable transparent windows systems like X-11 the graphical output of any application can be displayed on any other machine, which has an X-server installed. The X-Windows manager, which handles all operations like the movement of windows can run on a different machine in the network. The output of each application is hardware independent. An audio server should add the same functionality to audio applications as an X-Windows systems does.

For multimedia applications synchronization with other modalities is important. Some of the current implementation only allow to synchronize different audio streams like sampled audio data and MIDI streams (on SGI).

The experimental Network Audio System [7] has some of the requirements implemented,

but does not use 3D-sound. Synchronization mechanisms are not implemented yet.

The following sections will present the concept and the implementation of the Audio Application Server developed for the MIAMI-project. This server meets all the requirements mentioned above.

6.3.1 Concept

The Audio Application Server consists of three types of modules. The Audio Resource Manager, the input (ADC) and output server (DAC) and modules, which perform various signal processing operations. Applications which want to transmit audio data, send a message to the Audio Manager and request for a port to send their data to. If ports are available, the manager replies the address of the port to the client and an identification number, which will be used for referencing this audio stream, if parameters are modified in the future. Then the manager starts subtasks like the filters for the spatialization. If an audio output server on the desired host exists, the output channels of the filters are connected to the input of the output server, otherwise a new server is started first. If a client wants to change the parameters of the audio stream, e.g. direction, it sends a message to the manager, consisting of the command (*change direction*), parameters for the direction (*elevation, azimuth*) and the ID

The manager selects the new filter coefficients for the desired directions from the HRIR-catalogue and sends these coefficients to the filter tasks for this stream. If the application does not need the audio port any longer, a disconnect command will be sent to the manager and the tasks of this stream are disconnected from the audio output server and killed.

6.3.2 Implementation

Audio Resource Manager The Audio Resource Manager is a task starting all tasks and managing the links between each of this audio tasks. It also administrates the database with sounds (collisions, earcons, auditory icons) and the database with filter coefficients for the spatialisation of sounds.

Auditory subtasks There are several auditory subtasks like tasks like sources, signal processing tasks and output modules. The following tables give an overview of the implemented subtasks. This tasks allow to modify the properties of each audio stream independently. The gain of each channel can be changed and simple waveforms are generated online by software generators. Also sound files can be played. Currently a database

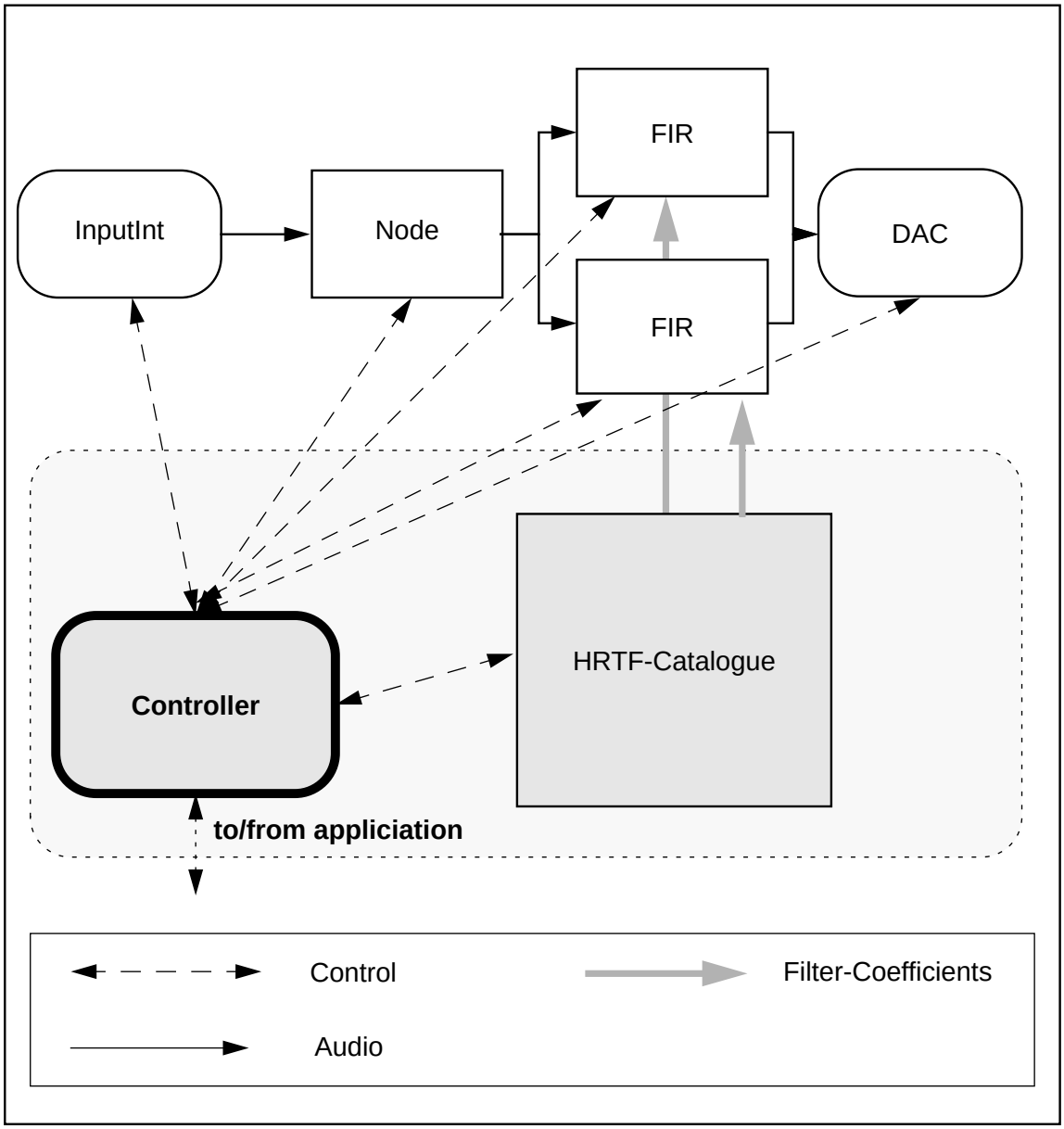


Figure 6.6: Structure of the Audio Application Server. Audio data stored in an integer file are read from the InputInt tasks transmitted to the FIR-filter via a Node and sent to the audio output server DAC

with simple auditory icons and another database with collision sounds can be accessed by the applications.

InputFloat	reading digital audiofile (float data)
InputInt	reading digital audiofile (integer data)
InputShort	reading digital audiofile (short integer data)

Table 6.1: Input objects of *AAS*

Pulse	generating a pulse-function
Rectangle	generating a rectangle-function
Sawtooth	generating a sawtooth-function
Sine	generating a sine-function
Noise	generating white noise

Table 6.2: Generator objects of *AAS*

adder	output is the sum of all input channels
fader	cos^2 -fading on one channel
fir	filter (finite impulse response filter) for one channel
iir	filter (infinite impulse response filter) for one channel
gaincontrol	sets gain for one channel
multiplier	output is the product of the input channels
node	sends the data from the input to multiple receivers

Table 6.3: Signal processing objects of *AAS*

Communication between subtasks All tasks mentioned above send or receive their data using *pvm_ostream* or *pvm_istream*. These functions are based on the message passing concept of PVM3. Each process is identified by a unique task identification number (task ID, TID). The sending process sends data to receiver, referenced by the TID. The receiver waits for messages of the sender with its TID. The processes may run on any machine in the PVM3-network. A *pvm_ostream* sends a package with audio data to a receiving subtask and pauses until the receiving tasks has started to process the data and sent an

acknowledge signal to the sender. This mechanism prevents to overrun the receiver with more data, which can be processed in time and allows a good balance of memory and processing power. Of course 'underruns' (no data available in the moment) due to problems of the network or overload of the CPU cannot be handled with such a mechanism. The length of a package is critical. Small packages allow a low latency of the system, but are very sensitive to problems during the transportation and cause a high overhead for packing and unpacking the data. Long packages reduce this overhead and make the system less sensitive to transportation problems but cause a high latency. Also synchronization with other modalities is affected, if the sampling or update rate is shorter than the duration of one package. In the current implementation each packages has a length of 20 or 40 ms, allowing update rates of 25 or 50 Hz.

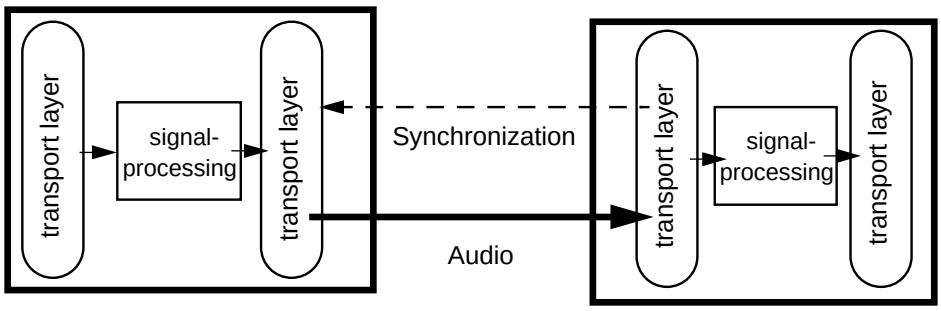


Figure 6.7: Message passing and inter-task synchronization

Spatialization The spatialization is realized by simulating the linear distortions, which change an acoustical signal transmitted from any position in space and received by the eardrum in the ear canal. The most important parameter are the interaural level differences, the interaural time differences and spectral changes for each ear signal. These effects are caused by shadowing and diffraction effects of the human body, the shoulder, the head and resonances in the pinna. The head-related impulse responses (HRIR) or head-related transfer function (HRTF) is one way for representing this characteristics. For the spatialization of a signal, each incoming sample has to be convolved with the impulse response of the desired direction. For a sampling rate of 44.1 kHz usually a filter-length of 60 to 70 coefficients is necessary. This demands for a a lot of computational power, which can be quantified by the convolution rate (CR). ($Convolution\ rate = sample\ rate * number\ of\ output\ channels * size\ of\ filter$). Reducing the sampling rate to 22.05 kHz also allows to downsample the head-related impulse response by the factor 2.

This leads to reduced CR, which is a quarter of the CR for 44.1 kHz. A further reduction of the number of coefficients might be achieved, if psychoacoustical properties are used to design more efficient filters [11].

Input and Output Server The main part of the audio input and output server is a hardware independent function. This part handles the communication with other audio modules and the addition of the audio data from different sources. Each time a package is received, the data are converted to the correct format (usually 16-bit Integer). For stereo output the data of the left and right channel are stored in interleaved order in a single buffer. The data are copied to the buffer of the hardware driver and the process is paused. The hardware reads the samples from this buffer and sends them to the digital-analog converter. If the number of samples, which have to read from the buffer reaches a 'low water mark', an interrupt is sent to wake up the server process and to start the reception of data and a refill of the the buffer again. For stereo input the interleaved data are separated and transmitted on different paths. The server calls low level functions to initialize the audio hardware, setting gain, sampling rate and the number of channels and to send or receive data from the hardware. These low level function call the hardware dependent functions. This approach allows to design the library for the intermediate level separately and adapt the server for any other architecture with little effort and risks, as the code for the server does not contain any hardware specific commands.

Synchronization Multimedia application also demand for the synchronization of multiple modalities. As audio has the highest sampling rate and the highest priority, the audio hardware will synchronize the other modalities. All tasks which have to be synchronized have to become a member of a PVM3-group (*sync-group*). Always when a package in the audio output server is processed, the current frame number is incremented. This frame number is then broadcasted to all other tasks of the *sync-group*.

6.3.3 Summary

The Audio Application Server (AAS) provides mechanisms and methods for transferring audio data between applications and multimedia terminals in networks. The routing is done automatically. Various outputs from different machines can be sent to the same output. Simple 3D-sound spatialization techniques are implemented. Distributed processing of complex audio signal processing chains can be performed using the processing power of the network instead of special DSP-hardware. AAS also takes advantage of multiprocessor platforms as independent task are exchanging data. Direct audio input from one machine

can be easily connected to audio output of another host. The Audio Application Server is designed for synchronizing other modalities with a resolution of 25 or 50 Hz. Application for the AAS range from videoconferencing (multiple inputs, each speaker at a different position in 'auditive' space) to the auditive presentation of non acoustical information by the means of earcons and auditory icons.

OutputFloat	writing digital audiofile (float data)
OutputShort	writing digital audiofile (short integer data)

Table 6.4: Output objects of *AAS*

6.4 The HARP Multimodal Environment

HARP is a multimodal environment designed for music, art, and entertainment applications which has been successfully applied in a number of real applications (Camurri 1996). The overall system architecture, sketched in Figure 6.4, is a distributed network of agents. The system is similar in some aspects to Cypher [22], TouringMachines [13], M [21], and NetNeg (Goldman et al. 1995). From a cognitive viewpoint, the system is structured in a long-term memory (LTM), the permanent, "encyclopedic" storage of general knowledge, and in a short-term memory (STM), the actual "context" regarding the state of the affairs of the world and the problems currently faced. Both LTM and STM are composed by symbolic and subsymbolic components, to face the different structure and nature of the domain knowledge. Following the scheme depicted in Figure 6.4a, we can make a distinction between the following basic building blocks that constitute the HARP architecture:

- Input Mapping: A group of agents able to receive signals from sensors and map them into perceptual spaces (e.g., a self-organizing map which classifies movement trajectories or sound signals).
- Output Mapping: A group of agents that manages the results coming from the cognitive processing agents, typically high-level parameters (e.g., the space of the "emotional" parameters for controlling the expression of an artificial face or the high-level timbral spaces of a section of a composition).
- Cognitive Processing: Two kinds of activity cooperate: (i) Subsymbolic agents perform subsymbolic planning and reasoning; (ii) Symbolic reasoning, which can only perform symbolic inference and planning in the symbolic database.
- Symbolic Database: it is a high-level, symbolic representation of the domain space(s) (e.g., music composition and performance spaces, movement and gesture spaces). It consists of (i) a symbolic knowledge representation language to serve as an interlingua for agents, and (ii) includes the ontology defining the terms used in the communication among agents (the symbolic LTM). It also includes a STM where relevant events, situations, objects, and related features are added during a work session. Agents are responsible to update such short-term DB, and can be triggered according to particular events occurred.

Another viewpoint of the HARP agents architecture is shown in Figure 6.4b, where the distinction between the two types of agents in HARP (experts and icons) is put into

evidence. As described in more detail in the next sections, experts are active, autonomous, skilled in a sub-set of the domain; icons are passive, providing to experts the access to (models of) the external world. The Symbolic Language The symbolic language allows to represent and manipulate the symbolic LTM and STM. The symbolic LTM consists of two components:

1. a terminological component appropriate for defining terms and for describing concepts and the taxonomic relationships between them. An inheritance semantic network formalism (Woods and Schmolze 1992) has been extended to represent and reason on time, actions and plans, and encompasses the possibility of using first-order axioms to extend its expressiveness.

2. an assertional component to represent factual long-term knowledge on the domain, based on first order logic. For example, the opening phrase of a well-known piece, say, Beethoven's fifth, is an assertional constant which can be considered part of the LTM. The assertional component can include factual generalizations expressed as first-order axioms (e.g. by means of quantified implications): for example, in the museal or theatrical scenario "all the moving objects on stage are humans".

The symbolic STM contains information concerning the specific events represented in its subsymbolic counterpart (e.g. sensory input). In other words, the STM symbolic component represents a single context of the actions represented in the subsymbolic component: it is a one-to-one symbolic representation of the entities and of the events in the subsymbolic component. The Ontology The symbolic level includes an ontology, i.e. the basic dictionary of the terms known and used by agents. Here, the concepts and axioms are introduced, which describe the basic ontological assumptions in our system. We do not assume the ontology we developed to be definitive or complete: it can be updated and modified according to possible new requirements of the domain. The concepts action and situation subsume all concepts that represent entities characterized by a duration or by a temporal location. Actions are the emerging representation in the symbolic database of expert agents; situations represent states of the world in which there are no significant changes, and are the emerging representation in the symbolic database of icon agents (see Figure 6.4b). The sub-concepts compound_situation and compound_action describe situations and actions which can be decomposed in terms of sub-parts, which are, in turn, situations and action, respectively. Roughly, this corresponds to agent societies in our model. An action produces some kind of change in the world: for this concept, the roles initial_situation, intermediate_situation and final_situation are defined. For each action, the filler of initial_situation is the state of the domain before the action is performed, while the filler of final_situation corresponds to the state of the domain after the action is performed. The fillers of intermediate_situation correspond to significant situations holding true during the performance of actions. While a generic action is, in general, simulative or a mere execution (evolutionary), a purposeful_action is an action characterized by a goal to be reached. The main difference between evolutionary and purposeful actions is mirrored by their inner agent structure: in the former case they are simple simulations or executions, in the latter case they can be characterized by planning capabilities to achieve the goal. The ontology here described has been further specialized for the developed system applications, with respect to gesture classification and music representation. Gesture taxonomies like those reviewed in [19][25] could be included in this ontology as sub-taxonomies starting from the concept situation.

6.4.1 The Subsymbolic Components

The STM symbolic knowledge base (KB) is linked one-to-one to the STM subsymbolic representation, which can either be connected to the world itself by sensors/actuators or to a mental model of the world. The LTM and STM subsymbolic components consist of a network of cooperative agents in the sense of (Steels 1994). In the subsymbolic level, agents

are concurrent applications in an object- oriented environment, each linked to terms in the symbolic components (see the HARP overall architecture sketched in Figure 6.4b).

6.4.2 HARP Agents: Experts, Icons, and Symbolic Agents

In HARP we distinguish between expert agents, icon agents, and symbolic agents (see Figure 6.4). An expert agent is skilled in a sub-set of the domain, in a given task, and is therefore capable to perform its task with a certain degree of autonomy and robustness. Icon agents embed representations of the external world or of mental models: therefore, icons provide sensors and effectors to experts. Icons encompass two aspects: firstly, diagrammatic or pictorial descriptions of situations (Chandrasekaran et al. 1993). These can be geometrical metaphors of a different domain: for example, several contemporary music notations used by composers (e.g. Kagel, Bussotti, Berio, Ligeti) are based on such kind of metaphors. The second aspect covered by icons is that they act as dynamic systems, as metaphors for reasoning on actions and plans. Landscapes of energy and models based on force fields are simple cases considered in this paper. We can see this kind of representation as an enrichment of the previous one, since it includes dynamics, force, and time coordinates within diagrammatic representations. Different experts that can perform navigation algorithms on N-dimensional maps have been defined in the subsymbolic component of the LTM. Force fields can be built by learning processes, as in the case of Leman's TCAD attractor dynamics system (Leman 1994) based on artificial neural networks. Expert and icon agents are formed by the following parts (Figure 6.4b):

- a body (the subsymbolic internal component of the agent, embedding its skill or the icon's internal representation);
- a (typically fast) communication channel for exchanging subsymbolic data among agents;
- pre- and post- codes, that embed all the communication between the agent and the symbolic database;
- a symbolic entity in the ontology in the symbolic database representing itself (the agent): an expert corresponds to a concept derived by action, an icon to a concept derived by situation. An agent is capable to signal possible intermediate situations occurred during its execution: this information can be used to cause the (de)activation of other agents.

A new instance of an agent can be generated by a new assertion of its symbolic entity in the database. The definitions of agents are resident in the LTM. To activate agents, they

must firstly be instantiated in the subsymbolic STM. Therefore, subsymbolic activity can only be carried out in the STM, i.e. in a completed context. The context can include the sonological level of music objects, as acquired by a model of the auditory system, instances of the transformation processes on such objects possibly depending on human movement/dance patterns. In a theatrical, "generalized choreography" model, the context can also include the agent internal representation and the three-dimensional world, possibly acquired by consulting its sensory input. Symbolic experts can operate solely on the symbolic database, i.e., perform internal cognitive processing. Symbolic icons play the role of recognizers: their task is to recognize particular world situations in the form they emerge in the symbolic database, and which may cause the (de)activation of some subsymbolic agents. HARP agents are supervised by the context manager, a module which takes care of activating/stopping agents and manages the symbolic-subsymbolic communication. Recognizers are scheduled according two main types of events: (i) the symbolic clock (low rate, see figure 1b), (ii) an event signaled by an expert. If any recognizer instantiates a new situation assertion, then the pre- codes will be executed to verify possible new firing of agents. Agents can cooperate in groups or societies to reach a common task: societies therefore act as a single compound entity, and this is reflected in the symbolic level, where societies are linked to a unique term (compound_action and compound_situation). For example, the agent acting as a Cicerone in a museum is formed in its turn by a number of agents for navigation, observation of the environment, multimodal user interaction. Both the Cicerone agent and all its sub-agents have symbolic counterparts. The generation and activation of a network of agents produces an execution, which consists of measurements and actions in the real environment, e.g., the external world, music and gesture spaces, a virtual computer-animated world, etc.

Symbolic vs. Subsymbolic Reasoning

The activity of experts and icons can be interpreted as a reasoning mechanism complementary to typical symbolic deductive systems: for example, navigation in a force field can substitute decision processes which would otherwise be difficult to model symbolically (e.g. with axioms or rules). The symbolic reasoning differs from subsymbolic reasoning in several aspects: (i) time granularity: subsymbolic reasoning is expected to react in real-time, since it has to follow and manipulate the flow of signals which usually require strict time constraints. Symbolic reasoning is usually expected to intervene at a much higher time granularity (seconds vs. milliseconds). For example, during the tracking of human movement, we might have a set of agents for input signal classification based on self-organizing networks. When a gesture is recognized then this is a "significant" instant for the symbolic level, a new instance of that gesture is added in the symbolic database, and this might cause the activation of symbolic agents which, for example, might try to

better classify that gesture, i.e., understand more deeply its meaning according to the current context and the historical data; (ii) the symbolic reasoning is carried out in both LTM and STM. In contrast, subsymbolic reasoning can only happen in a fully instantiated context, i.e., in the STM.

The context manager supervises and schedules the communication between agents: experts and icons are instantiated and connected to each other by the context manager, according to the specific context and the particular action/plan to be performed. The context manager defines and updates a set of input and output communication links for each expert and icon in the current context, complying with constraints in the symbolic memory (e.g. the topology of the semantic network and the current STM contents). The subsymbolic memory is therefore built and kept aligned with the symbolic memory during execution. Significantly, the context manager is able to manage and control the execution of hierarchical actions, i.e., actions recursively expanded into subactions by means of part of (or aggregate) relations, connected to input/output situations in hierarchies (in terms of both IS-A and part-of links). This corresponds to agent societies.

These basic HARP mechanisms are further explained in other chapters in the framework of the real world applications HARP/Vscope, HARP/DanceWeb, Theatrica/Museal Machine, and SoundCage. Implementation Issues HARP is written in C++ (Microsoft Visual) and Quintus Prolog, and run under Windows 95 and NT. Expert and icon agents communicate locally via OLE Automation. A Unix/Windows sockets library has been developed to connect agents running on remote Unix or Windows workstations. The Gateway module allows the communication with PVM agents. HARP applications can be developed and tested incrementally: for example, experts and icons can be developed and tested as stand-alone Windows applications: a few code constraints on their I/O must be followed to allow the HARP Agent Wizard module to automatically transform them into specific experts and icons integrated in a HARP KB. A work in progress regards the transformation of HARP into a software library, so that users can freely utilize the services of HARP from their applications.

Chapter 7

The Demonstrators

This chapter describes the current status of the work done on the implementation of both demonstrators in the MIAMI project: the symbolic demonstrator and the analogical demonstrator. Both demonstrators address the two major types of human-computer interaction, i.e., the handling of discrete, symbolical information, vs the handling of continuous 'physical' signals.

7.1 The Symbolic Demonstrator

This text describes the decisions made so far regarding the symbolic demonstrator of WP 4. At first, the contents of the Technical Annex will be outlined for your convenience, which shows the original design of this demonstrator:

WT 4.1.1

Objective

The adaptation of the general system architecture developed in WP 3 to the one used in symbolic demonstrator.

Approach

- Design of a graphics module for "Information City"
- Design of an information module for "Information City"
- Design of an output interface for navigation support and guidance

WT 4.1.2 Hardware adaptation

Objective

The adaptation of the hardware used by each partner in its domain for the symbolic demonstrator.

Approach

- Specification of minimum hardware requirements
- Provision of common software tools
- Ensuring for compatibility

WT 4.1.3 Software adaptation

Objective

To adapt and extend software used by each partner in WP 3 for symbolic demonstrator.

Approach

- Integration of specific software modules into one demonstrator
- Demonstrating of a prototype multimodal navigation through an "Information City"
-

Deliverable 5 Symbolic Demonstrator

Technical description The Symbolic Demonstrator has been chosen to show the benefits of multimodal integration in the context of information search and selection. The complete scenario consists of a "Information City" metaphore in which the user can navigate and interact with the system. The City metaphore maps information resources into a well-known city space with libraries, cinemas, casinos, etc. Due to the limitations in the available hardware when the trial implementation was done, the current implementation is restricted to the jackpot machine model around which one can navigate and perform selection of specific items with reasonable interaction speed. This is about the biggest model with which such interaction is possible currently. The extension to full model will be provided at later stage since more powerful hardware is now available. The demonstration will illustrate:

- Speed of the model operation
- Navigation and information selection

The exact form of the demonstration, an *integrated installation of software and hardware*, can only be described in more detail after the second year.

Form of presentation Demonstration of "Efficient Interaction and Manipulation Through Information Space Using Multimodality, Geometry and Time" to the Commission and the Public at a Workshop in combination with a review meeting

The demonstrator which will finally be presented will differ in the more complete implementation since suitable hardware will now become available.

7.1.1 Main features of the symbolic demonstrator

- The demonstrator will mainly consist of the Information City model
- It will include sound, visual and various input device modules

7.1.2 Contribution of the partners

Software and hardware which is already available or will be available in the near future will be used in this demonstrator:

Partner/TMM	Input	Output	General
RIIT (11)	partial (available) and full model (not yet available)	integration module	TkPVM
NICI (6)	pen software (not yet available)	—	TkPVM
DIST (6)	—	sound generation	concepts for a multiagent architecture
ICP (4)	audio components	talking/animated lips/face	—
RUB (3.5)	—	audio stream support	PVM streams & support

Table 7.1: The partners' contribution to the analogical demonstrator (TMM = Total Man Month)

7.2 The Analogical Demonstrator

In this section, we will describe the basic concepts and features of the analogical demonstrator of WP 4. At first, the contents of the technical annex will be repeated for your convenience. Then, the control concept, the system architecture, and the different operation modes of the *multimodal robot control station* will be described. Finally, the partners' contributions to the analogical demonstrator will be presented.

7.2.1 Technical description

WT 4.2.1 Adaptation of architecture

Objective

The adaptation of the general system architecture developed in WP 3 to a concrete one which can be used as a basis for the implementation of the analogical demonstrator.

Approach

- Designing the man-machine interface for a teleoperation station using several input and output channels
- Selecting the hardware for input and output of data and actions, taking into consideration the results of the experiments of WP 2
- Developing data models, structures, and types for the implementation of the demonstrator

WT 4.2.2 Hardware adaptation

Objective

The adaptation of the hardware used by each partner in their respective domains and used during WP 3 in this project to the analogical demonstrator.

Approach

- Integration of several hardware modules used for the bimodal experiments in WP 2
- Integration of different hardware modules used in the respective domains of each partner

WT 4.2.3 Software adaptation

Objective

The adaptation of the software used by each partner in their respective domains and the software developed during WP 3 in this project to the analogical demonstrator.

Approach

- Integration of several software modules and libraries into one demonstration system
- Realization of a prototype to demonstrate the advantages of a multimodal approach in teleoperation systems

Deliverable 6 Analogical Demonstrator

Technical description The Analogical Demonstrator has been chosen to show the benefits of multimodal integration in the context of object interaction and manipulation. The scenario consists of a teleoperation station in which the advantages of using more than one mode for the interaction with the system can be seen very clearly. The demonstration will cover among other things:

- Manipulation of virtual and real objects: multimodal expression and sound/music output in a virtual humanoid
- Telemanipulation and mobile robot guidance: development of a vocabulary of basic movements executable by a mobile robot, its integration with sound/music and facial outputs, and its use in a real context

The exact form of the demonstration, an *integrated installation of software and hardware*, can only be described in more detail after the second year.

Form of presentation Demonstration of "Efficient Interaction and Manipulation Through Information Space Using Multimodality, Geometry and Time" to the Commission and the Public at a Workshop in combination with a review meeting

For practical reasons, the demonstrator which will finally be presented will slightly differ from this description. The final setup will partly be build of integrated parts which are already available (see 6) or will be available in the near future in order to achieve the final goal of an integrated demonstrator.

7.2.2 Multimodal robot control

The control of a mobile vehicle (or robot) with advanced multimodal methods and sophisticated I/O devices is both, very attractive and useful. Take, e. g., the mobile platform PRIAMOS which is currently under development at the University of Karlsruhe. It is equipped with a multisensor system including 24 ultrasonic sensors, an active vision system (KASTOR), structured light, and a laser scanner.

Control concept

Its control concept follows the "shared autonomy" approach (see [12]), i. e. the robot receives its programs and data from a supervisory station and carries out its tasks autonomously. If an unforeseen event occurs, the robot asks the human operator for assistance. In addition, the operator can take over control at any time. Supervision of the

robot's tasks is performed with the sensory data sent to the operator and a simulation model which runs in parallel to or some time ahead of the real execution.

System architecture

These two components—a (teleoperated) mobile robot and a multimodal supervisory and control station—will be the 'heart' of the analogical demonstrator. For supervision and control of the mobile robot as well as the vision system, several different input devices/modalities which can be switched "on the fly" will be supported by using the META DEVICE DRIVER for data input (see 6.1). In order to provide sufficient feedback, several different output modalities will be supported by the system, e. g. force feedback with the FORCEJOYSTICK, acoustical feedback generated through the AUDIO APPLICATION SERVER (see 3.5), and a talking face for direct communication to the operator (*ICP-FACE-ANIM*, see 3.6).

The overall architecture for the supervision/control system will be developed based on the scheme depicted in fig. 7.1. It will have an input and an output layer, and several components will be realized as independent modules communicating via PVM, thus realizing some kind of multi agent architecture. This will make the cooperation between and the integration of the different modules supporting different modalities much easier.

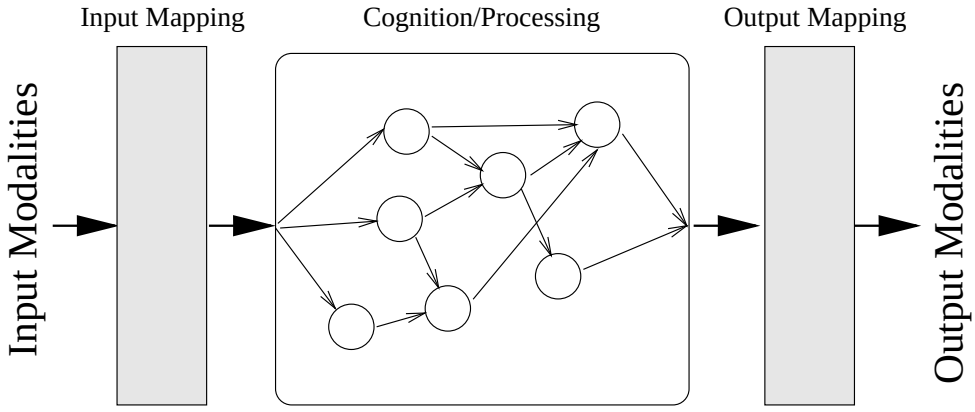


Figure 7.1: MultiAgent architecture (schematically)

Operation modes

In the final version of the analogical demonstrator, three different operation "modes" will be available:

Teleoperation mode Teleoperation of a mobile robot (as described above).

Error mode A mode in which intensive interaction is needed due to the robot's limited capabilities in unforeseen situations.

Simulation mode As no real manipulator is available on the mobile robot, a simulation with a combination of the mobile robot and a manipulator on-board can be used to demonstrate more features and especially interactions with other objects.

7.2.3 Partners' contribution

Software and hardware which is already available or will be available in the near future will be used in this demonstrator. In the following, the contribution of each partner with respect to the input modalities, output modalities, and general stuff is listed.

UKA (14 man month)

Input 6D mouse, FORCEMOUSE, FORCEJOYSTICK, other input devices, ultrasonic sensors, vision system, laser scanner

Output Simulation of two mobile robots (PRIAMOS and MORTIMER) in OpenInventor (see 1.1.4), shutter glasses

General *MDD* (see 6.1), PRIAMOS, MORTIMER, KASTOR

NICI (6 man month)

Input Pen, *GESTE* (see 6.2)

Output —

General TkPVM (see 1.2.1)

DIST (5 man month)

Input Exoskeleton

Output Manipulator simulation (in OpenInventor)

General Concepts for a multiagent architecture, *EXO* (see 3.8)

ICP (4 man month)

Input Lip reading, speech recognition

Output *ICP-FACE-ANIM* (see 3.6)

General —

RUB (3.5 man month)

Input Head tracker, microphone

Output "sound" (earcons, sonification, spatialization), loudspeaker, headphones

General *AAS* (see 3.5), pvm_streams

Bibliography

[1] V. Balasubramanian. State of the art review on hypermedia issues and applications. Worldwide web document, Graduate School of Management, Rutgers University, Newark, New Jersey, 1994.

[2] M. Bevan. Force-feedback technology. *VR NEWS - Virtual Reality Worldwide*, 4(6):23–29, July 1995.

[3] E. Bizzi. Central and peripheral mechanisms in motor control. In G. Stelmach and J. Requin, editors, *Advances in psychology 1: Tutorials in motor behavior*, pages 131–143. Amsterdam: North Holland, 1980.

[4] D. E. Broadbent. The magic number seven after fifteen years. In A. Kennedy and A. Wilkes, editors, *Studies in long term memory*, pages 3–18. London: Wiley, 1975.

[5] E. Burd, P. Chan, I. Duncan, M. Munro, and P. Young. Improving visual representations of code (submitted). In *International Conference on Software Maintenance, ICSM '96*, 1996.

[6] P. Dömel. Webmap - a graphical hypertext navigation tool. In *Proceedings of the 2nd International WWW Conference, Fall '94*, Chicago, USA., 1994.

[7] J. Fulton and G. Renda. The network audio system. *www: http://hospex.icm.edu.pl/pub/X11/contrib/audio/nas/ Network Computing Devices Inc.*, 1994.

[8] A. Geist et al. *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press, Cambridge, London, 1994. Also available via anonymous ftp from netlib2.cs.utk.edu as pvm3/book/pvm-book.ps; or via WWW at http://www.netlib.org./pvm3/book/pvm-book.html.

[9] D. Goldberg and C. Richardson. Touch-Typing with a Stylus. In *InterCHI '93 Conference Proceedings*, pages 80–87, Amsterdam, 1993.

[10] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. The unipen project of data exchange and recognizer benchmarks. In *Proceedings of the 12th ICPR*, pages 29–33, Jerusalem, Oct. 1994.

[11] K. Hartung and A. Raab. Efficient modeling of head-related transfer functions. In *Forum Acusticum - Proceedings, ACUSTICA united with acta acustica, Volume 82, Supplement 1*, page 88, Stuttgart Germany, April 1996. European Acoustics Association (EEIG), S. Hirzel Verlag.

[12] G. Hirzinger. Multisensory Shared Autonomy and Tele-Sensor-Programming – Key Issues in Space Robotics. *Journ. on Robotics and Autonomous Systems*, 11:141–162, 1993.

[13] I.A.Ferguson. Touringmachines: Autonomous agents with attitudes. *IEEE Computer*, 25(5), 1992.

[14] T. H. Massie and J. K. Salisbury. The PHANToM Haptic Interface: a Device for Probing Virtual Objects. In *Proc. of the ASME Winter Annual Meeting, Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Chicago, 1994.

[15] MIAMI. DI 2 - Progress Report. Internal report, ESPRIT PROJECT 8579 MIAMI, Apr. 1995.

[16] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity to process information. *Psychological Review*, 63:81–97, 1956.

[17] S. Münch and M. Stangenberg. Intelligent control for haptic displays. In *Proc. of the Eurographics '96 (to appear)*, Poitiers, France, Aug. 1996. INRIA.

[18] J. Neider, T. Davis, and M. Woo. *Open GL Programming Guide, Release 1.* Addison-Wesley Publishing Company, 1993.

[19] L. Nigay and J. Coutaz. A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion. In *INTERCHI'93 Conf. Proc.*, pages 172–178, Amsterdam, Apr. 1993. ACM, Addison-Wesley.

[20] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Company, 1994.

[21] D. Riecken, editor. *Special Issue on Intelligent Agents*, volume 37(7) of *Communications of the ACM.* 1994.

[22] R. Rowe. *Interactive music systems.* The MIT Press, Cambridge, MA, 1993.

[23] K. Salisbury et al. Haptic rendering: Programming touch interaction with virtual objects. In *Proc. of the Symp. on Interactive 3D Graphics*, pages 123–130, Monterey, CA, Apr. 1995. ACM.

[24] L. Schomaker and A. Thomassen. On the use and limitations of averaging handwriting signals. In . R. H. H.S.R. Kao, G.P. van Galen, editor, *Graphonomics: Contemporary research in handwriting*, pages 225–238. North-Holland, Amsterdam, 1986.

[25] L. R. B. Schomaker et al. A Taxonomy of Multimodal Interaction in the Human Information Processing System. Internal report, ESPRIT PROJECT 8579 MIAMI, Feb. 1995.

[26] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, New York, 1992.

[27] Slate, Lotus, GO, Microsoft, Apple, G. Magic, et al. *The Jot 1.0 Standard*. Software Publishers Association, 1993. Tel: +1-202-452-1600 ext 336, ftp://hplose.hpl.hp.com/pub/WWW/jot.html.

[28] P. Viviani and V. Terzuolo. Space-time invariance in learned motor skills. In G. Stelmach and J. Requin, editors, *Advances in psychology 1: Tutorials in motor behavior*, pages 525–533. North Holland, Amsterdam, 1980.

[29] J. Wernecke. *The Inventor C++ Reference Manual, Release 2*. Addison-Wesley Publishing Company, 1994.

[30] J. Wernecke. *The Inventor Mentor, Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*. Addison-Wesley Publishing Company, 1994.

[31] J. Wernecke. *The Inventor Toolmaker, Release 2*. Addison-Wesley Publishing Company, 1994.