Concept, implementation, and evaluation of a multimodal interaction style for music programming

IPO, Center for Research on User-System Interaction
PO Box 513, 5600 MB Eindhoven

19.10.1998

Rapport no. 1191

Concept, implementation, and evaluation of a multimodal interaction style for music programming

Steffen Pauws

Voor akkoord:
Dr.ir. F.J.J. Blommaert

b.a. Ing. J.J.C.M. Ruis

Concept, Implementation, and Evaluation of a Multimodal Interaction Style for Music Programming

Steffen Pauws

Date of issue: 10/98

Title	Concept, Implementation, and Evaluation of a Multimodal Interaction Style for Music Programming.
Author	Steffen Pauws (IPO).
Reviewers	Don Bouwhuis (IPO), Berry Eggen (USIT/IST), Dik Hermes (IPO), Erik Moll (USIT/IST).
Project	Accessing large amounts of information in multimedia applications for home entertainment environments aka Adaptive Multimodal Interaction (AMI).
Funding	Philips Research Nat.Lab. USIT/IST (group Collier)
Keywords	multimodal interaction, software design and implementation, usability evaluation, music selection and programming, nonvisual interaction, learnability, mental model, tactual feedback by force feedback, auditory feedback by non-speech audio and speech synthesis.

Abstract New media and portable devices afford to select and program favourite music from a large music collection while one is on the move. However, contexts-of-use outdoors are typically characterized by impoverished or even lacking visual display of information. Moreover, visual inspection while listening to music may not be pleasant; a music listener may want to devote complete attention to the music. If visual display of information is inadequate or undesired, other nonvisual sensory modalities have to compensate for that. This report describes the design, software implementation, and usability evaluation of a multimodal interaction style for music programming. By means of user involvement, the conceptual model of the interaction style evolved into a visual roller metaphor resembling a slot machine. In the style, user control of the interaction proceeds entirely by manipulating the IPO force feedback trackball (Engel, Haakma, van Itegem, 1990). Tactual feedback, mediated by the trackball, mimics the feel of setting rollers into motion. Non-speech audio, generated by software, imitates audible mechanical clicks of roller movements. Speech synthesis gives status information of particular states of the interaction. The evaluation investigated the level of efficiency and the learning of procedures of the interaction style, in particular, in the presence and absence of visual display of information. The participant's task was to compile a music programme as quickly as possible. Task performance was measured by compilation time and number of actions. Score on procedural knowledge was assessed by a post-task questionnaire. It was observed that participants performed equally efficiently, i.e., not significantly different, in both visual display conditions, except for the first programming task. In the first task, nonvisual interaction required significantly more time (approximately one additional minute) and more, but not significantly more, actions, probably due to exploring the interaction style for developing a mental model of the interaction style. Foreknowledge about the visual display is not strictly necessay; at least, it does not improve task performance in the absence of a visual dis-play. The results of the evaluation demonstrate that tactual and auditory feedback can compensate for the visual modality in contexts-of-use in which visual display of information is impoverished or even lacking: portable devices, remote control, car equipment. Continuing research effort in applying tactual feedback, e.g., by means of force feedback devices, and auditory feedback, e.g., speech synthesis and non-speech audio, in interaction concepts in user interfaces is justified.

Contents

Executive summary

A multimodal interaction style for music programming has been designed, implemented, and evaluated on usability. The challenge during design was threefold: minimizing the number of control elements without sacrificing usability, facilitating contexts-of-use with and without a visual display, and devising an interaction style in which music recommendations are provided.

The conceptual model of the interaction style comprises four rollers that can be manipulated by a force feedback trackball. Each roller represents a concept in the music programming domain; so, there is a roller for the music collection, the music recommendations, the music programme, and the music styles. The pieces of music, which are represented by the title and performing artist, are wrapped around a roller. By lateral roll movements of the trackball, the appropriate roller can be selected. By forward and backward roll movements, the appropriate item can be put at the front of the roller. Now, by pressing the ball, pieces of music can be added to the music programme, or removed from it.

The presented multimodal interaction style consists of a manual input modality mediated by the IPO force feedback trackball and three output modalities: the visual, tactual, and auditory modality. The visual modality consists of a graphical and animated representation of the roller metaphor; they show a jerky motion, when turned gradually, and a continuous movement, when stroke more vigorously. The tactual modality is addressed by force feedback mediated by the trackball; it imitates the feel of friction and compliance parts of roll movements. The auditory modality is addressed by non-speech audio and speech feedback. Non-speech audio mimics the quirky sounds of rollers; it is generated by software. Speech feedback gives some status information on the interaction; it consists of the concatenation of pre-synthesized phrases.

Complete description of the design process, design decisions, and software is provided in this document.

The purpose of the experimental evaluation was to assess the usability properties of the presented interaction style. In particular, the experiment investigated the level of efficiency, i.e., time and number of actions, and the learning of procedures, i.e., score on post-task questionnaire, while performing music programming in the presence and absence of visual display of information. The experiment demonstrated that users were able to acquire such a proficiency after three minutes of free exploration and without procedural instructions that they could complete a given music programming task effectively in both visual display conditions. In addition, they learned to perform the tasks more efficiently in both conditions; they needed less time and fewer actions for each successive task.

Though nonvisual interaction indeed involves a considerable cognitive load, users who worked without a visual display were able to perform as efficiently as, i.e., not significantly differently than, users who worked with a visual display, already after the first task. The cognitive load is mainly determined by the procedures that have to be explicitly revealed and memorized by the user while interacting nonvisually. A first task without a visual display is probably devoted to revealing procedures and incorporating them in a mental model of the interaction style. Foreknowledge about the visual display is not strictly necessary; at least, it does not improve task performance.

It is demonstrated that the tactual and auditory modality can compensate for the visual modality in contexts-of-use in which visual display of information is impoverished or even lacking: portable devices, remote controls, car equipment. The results justify a continuing research effort in applying tactual feedback, e.g., by force feedback devices, and auditory feedback, e.g., speech synthesis and auditory icons, in interaction concepts for user interfaces.

1 Introduction

This document describes the concept, design, implementation, and usability evaluation of a multimodal interaction style for music programming. The interaction style is a proof-of-concept prototype reflecting the design decisions made in the preparation of the conceptual model, and not the design decisions that are driven by implementation or hardware concerns ('prototype' definition by Harel (1992)).

The software aspects of the interaction style are addressed in Chapters 2 through 9; the content of these chapters is summarized in Section 1.2. The usability evaluation of the interaction style is described in Chapter 10.

1.1 Funding

This document reports the results of the concluding activities of the project which is formally administered under the name 'Accessing large amounts of information in multimedia applications for home entertainment environments', but often informally referred to as 'Turn on the Base (ToB)' or 'Adaptive Multimodal Interaction (AMI)'. The project is based on a funding of Philips Research Nat.Lab. USIT/IST (group Collier).

The concept formulation, design, and implementation of the interaction style were carried out during a 5 month visit at the USIT/IST group. The experiment and completion of this document were done at the IPO (Centre for Research on User System Interaction).

1.2 Software overview

The interaction style is targeted on the Microsoft Windows 98 platform. Therefore, a brief summary of Microsoft technology is given in Section 1.2.1. The software is described in Section 1.2.2.

1.2.1 Microsoft technology

It is hard these days to decipher to what comprehensible notions and interconnections acronyms such as COM, ActiveX, and DirectX refer; are they products, technologies, standards, brand names, or marketing strategies of Microsoft? It also seems that the names stay, but to what it applies varies over time or vice versa. For any programmer, novice to the Microsoft development tools, it is rather frightened what steep learning curve booms up when he or she wants to familiarize even the rudiments of Microsoft Windows programming. Any Microsoft product is nowadays permeated by the Component Object Model (COM) technology, so it is recommended to acquire some understanding of COM's underpinning. A highly condensed description of COM, ActiveX, and DirectX follows in the next three sections.

1.2.1.1 COM

The Component Object Model (COM) is a specification standard which defines how software objects or components interact with one another (Rogerson, 1997). An object which complies to the COM standard is called a COM object. Compliance is simply established by inheriting the so-called *IUnknown* interface and implementing the three declared functions from that interface: *QueryInterface*, *AddRef*, and *Release*. Clients, that have a reference to an *IUnknown* interface, know that they have an 'unknown interface' for an implementation of an object, but also know that they are able to ask for the specification of that interface by calling *QueryInterface*. As multiple clients might have references to the same COM object, COM objects cannot be destroyed at will. A COM object has the responsibility to destroy itself when it 'feels' it is no longer needed. An object's life span is managed by maintaining the number of times it is referred to. An internal reference count is decreased when the interface of a COM object is queried by a client. A client is obliged to explicitly notify the COM object when it makes no longer use of the interface by calling *Release*, which decrements the reference count. Whenever the client makes a copy of the referred interface, it should also notify that fact to the COM object by calling *AddRef*, which increments the reference count.

Knowing what three functions distinguish COM objects from other objects and what obligations there are for clients to use interfaces of COM objects, we now come to the point what privileges the COM standard actually grants. Essentially, COM facilitates the development of component-based software architectures, in which components can be easily replaced by others and reused in other contexts. In this respect, the component is a coherent functional package in binary code, i.e., a mini-application, which connects with other components at run time to form an full-fledged application. Snapping pieces together in a flexible way puts heavy demands on software properties such as version control of interfaces, polymorphism, dynamic linking, and language independence. Components written to the COM standard fulfil these requirements; they come as Win32 dynamic link libraries (DLL) or as executables (EXEs). In addition, Distributed COM (DCOM) facilitates the interaction of remote components over a network including security mechanism for authentication and encryption.

1.2.1.2 ActiveX

It all started with *object linking and embedding* (OLE) which viewed the desktop metaphor as document-centric. The world was made out of compound documents in which for instance, spread sheet data could be manipulated within word processors. The first version of OLE was based on dynamic data exchange (DDE), a now obsolete data-transfer technique. The second version of OLE disregarded DDE and introduced Component Object Model (COM) as its new foundation. As such, OLE was the first COM-based product. For some while, OLE was tagged on any product or technology that had COM ingredients. As the acronym OLE no longer stood solely for compound documents, the name ActiveX was chosen to refer to the loosely defined set of COM-based technologies.

An ActiveX control is a COM object, i.e., a mini-application in the component-based software architecture. Originally, its functionality was tailored for the desktop, but that is no longer a necessity as it also applies to the domain of the internet. An ActiveX con-

trol is implemented as a dynamic link library (DLL) and the location of this DLL can be looked up in the Registry. The Registry is the shared database of component locations on Windows platforms. An ActiveX control needs to be embedded in a run-time environment such as Visual Basic or a web browser to run. The methods of an ActiveX control are provided to Visual Basic client applications via *Automation*. Automation is the reserved name for the way methods of COM objects are exposed by their interfaces to interpretive languages such as Visual Basic. Behind this standardized interface, an ActiveX control is allowed to virtually do anything.

1.2.1.3 DirectX

In contrast with ActiveX, the DirectX technology is rather an *application programming interface* (API). It is an API for multimedia application programming on Windows platforms. Different parts of DirectX provide interfaces for different multimedia domains such as

stereo sound and sound in three dimensions (DirectSound),

rendering of simple graphics, animation by colour palettes (DirectDraw),

rendering of complex graphics (Direct3D),

input from devices and output to force-feedback devices (DirectInput), and

communication for multi-player network games (DirectPlay).

The near future promises that the DirectX API will be supported on any Windows platform. It is marketed as a product that provides almost asynchronous low-level access to multi media hardware in a device independent way without delay latency. All DirectX object classes provide a functional set of methods on a variety of multi media platforms, which interfaces are compliant to COM. If the underlying multi media hardware has the appropriate capabilities, the implementation of these methods will directly call on the hardware known as hardware acceleration. Otherwise, the methods will be emulated in software and system memory (Bargen and Donnelly, 1998).

1.2.2 Software components

The presented interaction style is implemented in Microsoft Visual Basic 5.0. It comprises several components, i.e. ActiveX controls, as shown in Figure 1. All software components including the trackball device and the music database are described in separate chapters. As the interaction style utilizes only pre-synthesized speech from an external resource, no separate speech synthesis component was considered necessary to meet the specific objectives of this project. Play back facilities for speech are implemented in the non-speech sound generation component. Also, two other software components are not addressed in this document, viz. SelectionWheel which is only a limited version of SelectionBall, and ShadedRegion which renders a smooth curved surface by using an illumination technique. The SelectionWheel component was considered too similar to the SelectionBall for devoting a complete chapter to it. The functionality of the ShadedRegion component was considered too minimal.

First, the interaction style itself is described in Chapter 2. However, to acquire a complete overview and comprehension, the reader might need to jump back and forth in the chapters. The music database design and content is described in Chapter 3. Some of the mechanics and electronics of the IPO trackball with force feedback are highlighted in Chapter 4. The control software of the trackball comprising three layered components (Lighthole, TacServer, and TacServer Extension) is documented in Chapter 5. The audio output part is described in three separate chapters, starting with the lowest level of streaming audio (AudioDevice) in Chapter 6, followed by generating non-speech audio in Chapter 7, and concluding with decoding MPEG audio in Chapter 8. The visualization of a roller by the SelectionBall component is described in Chapter 9.

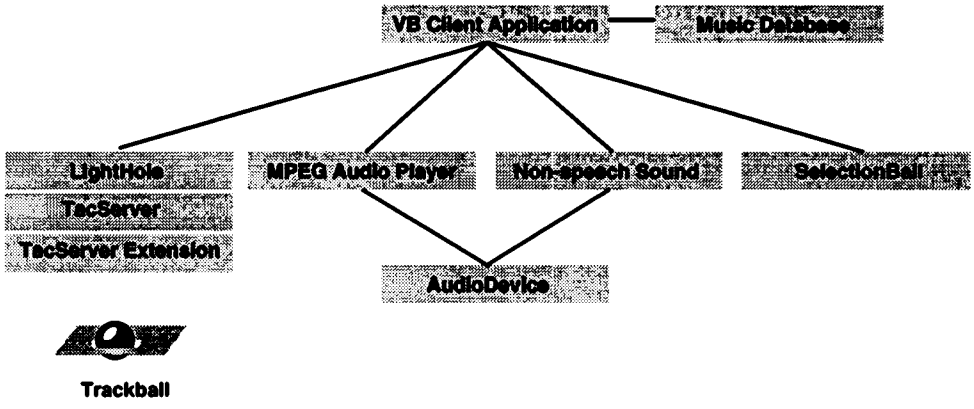

Figure 1. The component-based architecture of the interaction style.

As might be clear from the considerable number of chapters, we have attempted to be as complete as possible. Not all software components are implemented 'from scratch' within this project; we adopted a clever 'borrow management' of the deliverables from other, but like-minded projects, and adapted the components to meet our needs. However, the components did not excel by level of documentation. In order to repair this tiny imperfection from the past, to save our effort in survey research and reverse engineering for those that will follow us, and to neatly compile the making of a multimodal interaction style, we decided to make a complete reference to all software. The first section of each chapter deals with the design decision of a software component. Subsequently, the chapter shades off into the implementation aspects of the software component. Each chapter concludes with the interface of the component, i.e., how the component can be integrated in a client application.

2 A multimodal interaction style for music programming

Programming one's favourite music is probably best described as an iterative search to find multiple music options best suited for the intended purpose. Music listeners adopt a particular choice strategy that incorporates both global aspects, e.g., eliminating all items from further consideration that do not comply a pre-defined standard, and local aspects, e.g., pair-wise comparison to resolve particular trade-offs in choice. At the outset of the project, three main requirements were stated to be met by the interaction style.

Minimal effort for the user to program music. Although dealing with music entails an intrinsic enjoyment for many of us, music programming is often difficult. For instance, music listeners must evolve an idea what musical content is available from the collection, and must accommodate their choice strategy accordingly. As choice strategies are intertwined with global and local aspects, an interactive system should not further complicate the music programming task. The user of an interactive system must experience full control over the applied strategies. The user must be able to adequately monitor the progress of the music programming task, while spending minimal effort to attain intended results and to learn how to operate the interactive system.

Facilitate different contexts-of-use while programming music. There is no obligation to do a music programming task solely at home. Surely by the advent of new media and portable devices, music programming might equally well be performed outdoors, for instance, when one is on the move. Whereas visual displays are likely to be available at home, a portable or wearable device is far harder to equip with a qualitatively acceptable monitor at a low price. A visual display requires visual inspection of information which may be unpleasant in a music programming task. Also, the given context-of-use may prevent the listener from using a visual display, because there is no visual display available or it is already used by others. To facilitate the context-of-use when no visual display is used, feedback of information should not solely pass the human visual channel, but also the tactual or auditory channel have to be employed, perhaps equally well, for that means.

Provision of music recommendations while programming. Previous studies showed that an automatic music compilation facility such as PATS (Personalized Automatic Track Selection) (Pauws and Eggen, 1996; Eggen and Pauws, 1997) is a feature that enables faster selection of preferred music, or, at least, is considered a welcomed variety to the music programming task (Pauws, Ober, Eggen, and Bouwhuis, 1996; Pauws, Eggen, and Bouwhuis, 1997). In this respect, recommendations are defined as music options that are suggested by the interactive system to be included in the music programme. What recommendations are presented is however beyond direct control of the user. The interaction style must provide a coherent framework in which the music is recommended while the user is doing its programming task.

The proposed interaction style for music programming is considered a multimodal interaction style in the sense that additional sensory ways of information display, control, and feedback are utilized. Multimodal interaction enables the user of an interactive system to explicitly employ different (sensory) modalities to perceive and convey information and to execute actions. If the choices of communication modalities are well coor-

dinated, it is often suggested that the attention to different facets of a task can be easily divided amongst different modalities. Consequently, following the same line of thought, interactive systems will be more usable and even more pleasurable to use.

It is demonstrated that the addition of tactual cues in simple motor control tasks improves task performance. For instance, Nelson, McCandlish, and Douglas (1990) showed that vibratory feedback additional to visual feedback reduces response times in reaction time experiments. There is also evidence that tactile information leads to increased velocity in finger movements, from which it is suggested that it reduces visual load in completing tasks (Akamutsu, 1991). In studies involving the use of a mouse or trackball with force feedback, movement times in target acquisition tasks were shorter when tactual and visual feedback were provided than when only visual feedback was provided (Akamatsu, and Sato, 1994; Engel, Goossens and Haakma, 1994; Akamatsu and MacKenzie, 1996). Akamatsu, MacKenzie, and Hasbrouc (1995) reported that final cursor positioning times were shorter for conditions with tactile feedback, and to a lesser extent for conditions with auditory and colour feedback, than a normal condition without additional feedback in a target acquisition tasks with a mouse. It is however remarkable that only a few systematic usability studies are documented that addresses the potential usability of multimodal interaction styles in complete interaction concepts (see e.g. (Gourdol, Nigay, Salber, and Coutaz, 1992; Hauptmann and McAvinney, 1993; Robbe, Carbonell, and Valot, 1997) for evaluations on the combined use of speech and gestures/pointing).

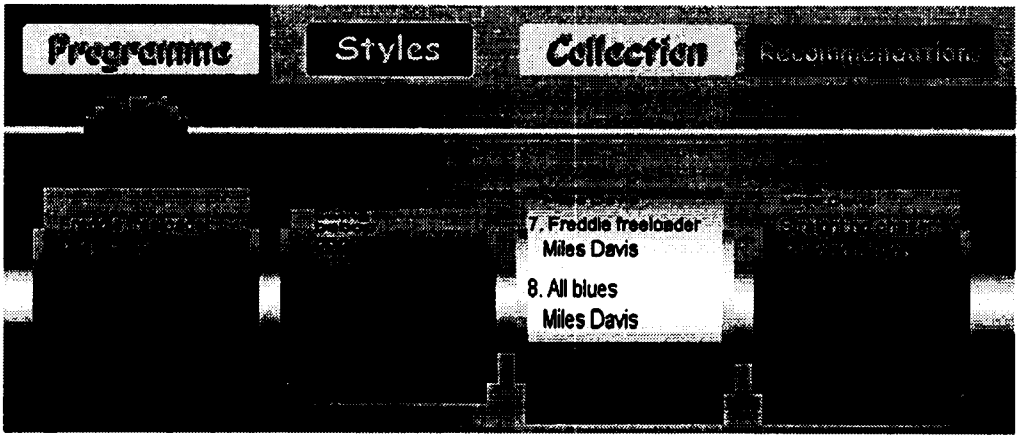

Figure 2. The visual display of the interface.

As shown in Figure 2, the visual display of the interaction style resembles the physical features of a slot machine. The interaction style displays four main rollers, on which the music programme, the music styles, the music collection, and the music recommendations are projected. In the style, user control of the interaction proceeds entirely by manipulating the IPO force feedback trackball (Engel, Haakma, and van Itegem, 1990). The force feedback trackball was used to minimize the number of control elements without sacrificing issues on effectivity and efficiency. Essentially, the trackball can be moved in two dimensions, i.e., in lateral and back/forward directions. In addition, the trackball can be pressed. By backward and forward trackball movements, rollers can be set in motion to bring another item at the front. A small hand movement brings the next

item to the front of the roller. A somewhat faster hand-stroke covering a somewhat longer distance skips the next two items. By lateral trackball movements, one can hop from one roller to another roller. The tactual force feedback mediated by the trackball conveys the feeling of setting rollers into motion or jumping from one roller to the other; for instance, the discrete steps of a roller movement feel like bulges and notches while carrying out the movement. By using the roll movements, a user can choose a particular music style on the music styles roller and search further for pieces of music within that style on the music collection and recommendation rollers. A single press on the trackball, i.e. single click, evokes information about the current music style to be uttered. Double-pressing the trackball, i.e., double-clicking, means adding or removing a piece of music to or from the music programme.

Rotating the roller step-wise also produces audible clicks. Larger roller movements produce a rattling sound. Synthetic speech feedback informs the user about states of the interaction; the user is informed about which roller is currently in focus, how many pieces of music are added to the music programme, what music style is selected, etc.

2.1 Design process

The design process followed a strategy in which a given conceptual model for interaction was extended by adding required user actions and functions one-by-one. The conceptual model was refined or completely revised when adding a function implied a conceptual 'breakdown' or an inconsistency of operation. For instance, a conceptual 'breakdown' occurred when the addition of a function required user actions that had no direct meaning with the conceptual model. To accomplish a high level of learnability, operating the interaction style should be consistent. Consistency of operation means that the same pattern of actions can be used in different situations, allowing the user to learn such a pattern only once. As shown in Figure 3, several metaphors as a conceptual model were considered, before arriving at the final one as shown in Figure 2.

It must be emphasized that the design process was not a pure top-down process. Some constraints pertaining to restrictions on resources, capacity, and implementation feasibility also influenced the design decisions.

Minimizing the number of control elements without sacrificing efficiency and effectivity is assumed to improve learnability of an interaction style; the user then has to learn the meaning of fewer terminals, e.g., button presses, of the interaction style. An expressive interaction language is required to minimize the number of terminals. A force feedback trackball is regarded a device with high expressive power; only one control element, i.e., the trackball itself, is present on which a repertoire of actions can be formed (see also Section 2.1.1).

A 'natural' conceptual model in conjunction with the trackball is to visually represent items, i.e., pieces of music, on a spherical object (see A in Figure 3). A roll movement of the trackball then directly corresponds with a proportional rotation of the spherical object. However, despite its intuitive appeal, rotating the sphere to put a particular item at the front means also a change of the context, i.e., the locations of all surrounding items change.

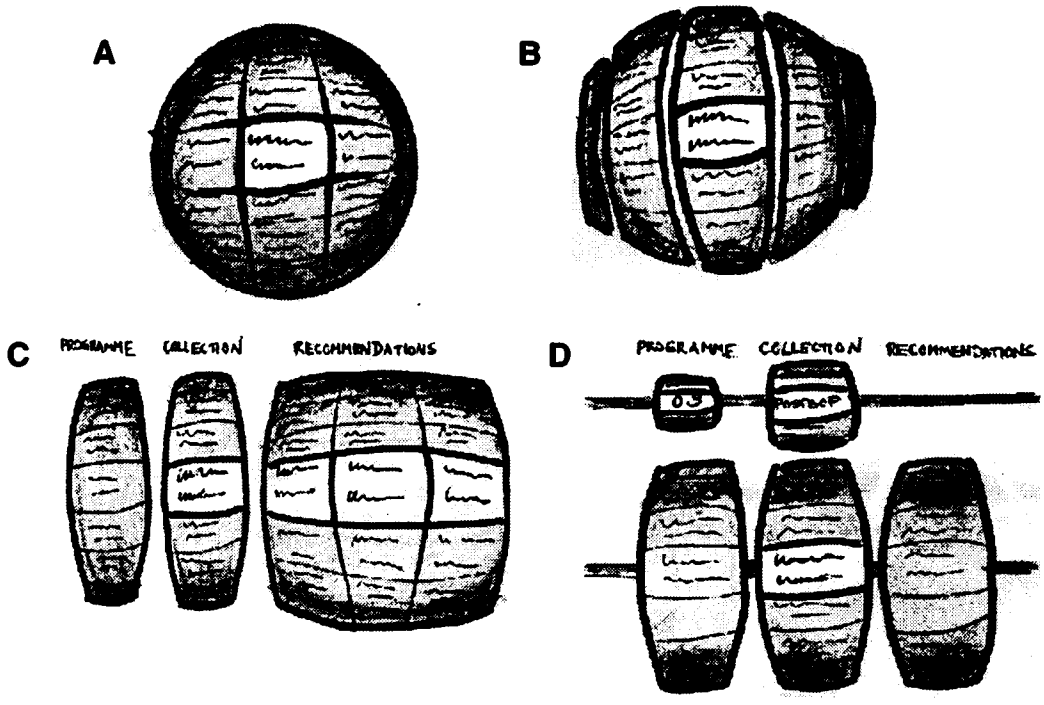

Figure 3. Previous conceptual models for interaction.

To avoid this recurring context change, the sphere was cut into slices. Each slice can then be rotated independently (see B in Figure 3). However, it was concluded that a sliced spherical object does not easily allow the representation of concepts in the music programming domain.

By assigning each slice to a particular concept in the music programming domain, one can jump back and forth from music programme, music collection, and recommendations by lateral roll movements of the trackball (see C in Figure 3). The slices evolved into rollers. Each piece of music in the music collection contained a list of three recommendations which were presented on a big roller in a horizontal fashion. However, operation of the three rollers was inconsistent. Whereas backward and forward roll movements of the trackball were required to roll to the next item on the programme and collection roller, lateral roll movements were required to move across the items on the recommendation roller. Backward and forward roll movements had even no effect on the recommendation roller.

The obvious refinement was the use of three rollers with the same action characteristics (see D in Figure 3). Also, a counter that indicated the number of pieces of music in the music programme was added. To accommodate the selection of music styles, an additional roller was placed above the collection roller. Conceptual model D in Figure 3 was implemented in a working prototype and worked out with user involvement; the previous concepts were only discussed with colleagues at the laboratory.

In an iterative fashion, two different users worked with the interaction style, and imperfections pertaining to usability were subsequently repaired. If neither user was able to acquire such proficiency after five minutes of free exploration that he/she could perform a given music programming task effectively, it was concluded that learnability was insufficient. Users spontaneously made suggestions for improvement. No more than five iterations, i.e., 10 users, were necessary to master the interaction style; also, no more important shortcomings or user complaints were reported. It was concluded that no further refinements were considered necessary. The final interaction style is shown in Figure 2. The prime imperfection was the location of the music styles roller; it could not be manipulated directly, but only by 'vigorous; roll movements when on the collection roller. Again, this was an issue of inconsistency which was easily resolved by placing the music styles roller at the level of the three main rollers.

It may be clear that the design process was primarily dominated by the development of a visual metaphor,. Tactual and auditory feedback were designed to be in use of this metaphor. Force fields were parameterized and laid out to convey the feel of setting mechanical rollers in to motion. Non-speech sound was designed to mimic the impact sound of a object made of a natural material (e.g., metal, wood, glass, rubber) been hit by another object. Pre-recorded samples from four different speech synthesis applications were initially incorporated in the interaction style. Only one was retained on the basis of its clear intelligibility. The quality of the tactual and auditory feedback were informally evaluated in close cooperation with colleagues at the laboratory.

2.1.1 Interpretation of trackball actions

The actions originated from a trackball device can, in principle, be interpreted in two distinct ways (see Figure 4). The actions can be absolutely mapped onto a cursor position, i.e. a hot spot, or, the actions can be directly mapped onto the objects of interest, i.e., the rollers in our case.

- *Absolute mapping on location.* The most common way is to consider the trackball just as the way it is, as an absolute pointing device. The ball rotation movement is mapped in an *absolute* way to the cursor movement. This absolute mapping gives problems when the cursor tends to go 'off-screen'. Obviously, the trackball can not be taken from its housing and placed somewhere else as easily as a mouse can be tilted from its mouse-mat.

  If, so to say, the trackball is integrated into the well-known desktop metaphor without further ado, it reveals some inherent limitations. For instance, the cursor has only a localized impact. As the regions of action are scattered in a workspace, there is only a single cursor that has to do all the work. A trackball device is, by that, a notoriously *inefficient* apparatus to perform target acquisition tasks by its absolute mapping nature (Douglas and Mithal, 1997), which makes moving the cursor back and forth in a complex navigation task difficult.

  Another issue opens up when we consider nonvisual navigation. The crux of nonvisual navigation is based on abilities to encode cues from other sensory modalities about the workspace, to acquire an accurate and persistent mental representation of the workspace, and to align and add new cues to this representation by inference mechanisms (see also Section 10.2.2). By laying out these cues as landmarks in a two-dimensional space and providing only a single probe (as is the case in the desktop

metaphor), it is likely that a user will get lost in navigational space. A user does not only need to remember what cues or objects are relevant for a given task, but also where, with respect to some current cursor position, these cues are located.

- *Direct mapping on objects.* The actions originating from the trackball can also be directly mapped on the objects of interest. As such, rolling the ball has an immediate incremental impact on the representation of the object in focus and has, thus, direct meaning. As actions simply happen in the world and their effects can be immediately perceived, it is proposed that this 'direct engagement' further shrinks the cognitive effort to execute actions purposefully (Hutchins, 1989). It emphasizes causality in the interaction style and may improve usability. Although the concept of a cursor is still required for keeping track of the ball position behind the scenes, it is hidden from the user's perception. As a result, the user might experience the trackball device merely as a means to execute gestures, i.e., instantaneous hand-finger movements, than a means to point at things. Additionally, the user does not have to be aware of a current cursor position and what task-relevant objects are nearby in navigational space, which facilitates nonvisual navigation.

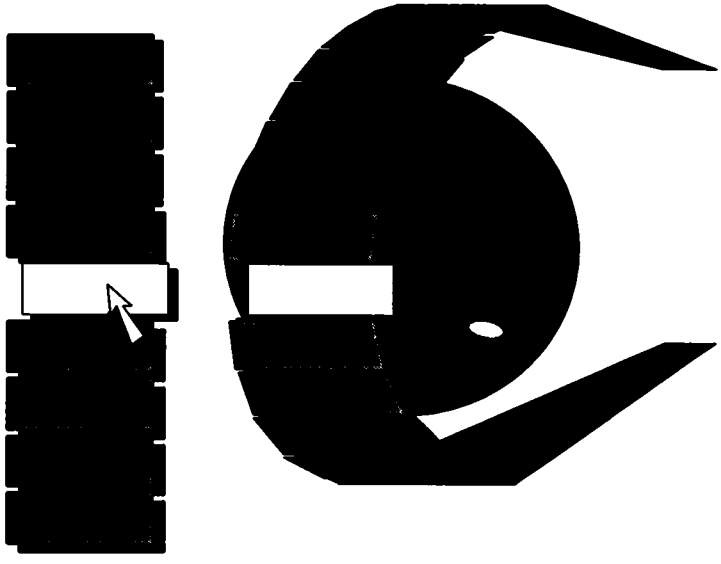

Figure 4. Two interpretations of trackball actions. The left-hand figure represents the absolute mapping interpretation. The right-hand figure represents the direct mapping interpretation.

In the interaction style, the trackball device is used to mediate actions directly on the objects of interest (as shown in the right-hand figure in Figure 4). The objects of interest in the proposed interaction style are rollers. The roller in focus is made salient by highlighting its visual representation. To enforce a 'direct engagement' with the rollers, the trackball's freedom of movement is constrained such that it is not possible to move the ball diagonally. This is done by an appropriate lay out of force fields, i.e., tactual objects, as shown in Figure 5. Rectangular force fields, feeling like small gutters, are placed in a cross. This cross captures the ball and guides the movements of the ball along straight lines. A global circular force field, i.e., a hole, surrounds this cross and slightly pushes the ball towards the centre of the force field; it requires some hand-force for leaving its

centre. Force fields called bumps that feel like small raised borders are placed at the ends of the cross. Moving across the bumps, the ball will be captured by small holes. The configuration of a bump followed by a hole gives the felt sensation of a mechanical click. When the ball is captured by one of these terminal holes, the roll movement is considered to be finished. Immediately, the ball is positioned at the centre of the cross. As shown in Figure 5, two additional terminal holes are placed at the vertical axis of the cross. Their function will be clarified later on.

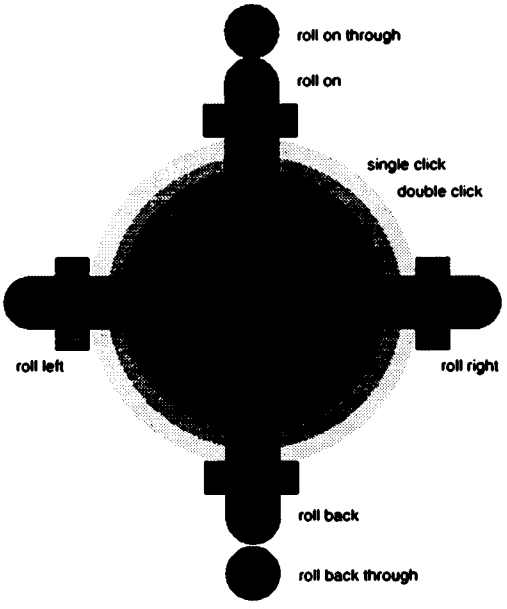

Figure 5. The force fields are laid out such that the ball can only be moved along straight lines.

In total, eight user input events are defined to originate from the trackball device. As shown in Figure 5, six user input events have to do with roll movements: roll_left, roll_right, roll_on, roll_down, roll_on_through, and roll_down_through. The other two events occur when pushing or double-pushing the ball; they are denoted respectively as click and doubleclick. Rolling the ball to the left or to the right,i.e., lateral roll movements, corresponds to going from one roller to the other. Rolling the ball forward or backward means rotating the roller itself.

The following two sections deal with the formal specification of the behavioural aspects of the interaction style and the multimodal feedback.

2.1.2 Statecharts

The behaviour of the interaction style on the user input events is formally specified by statecharts (Harel, 1987) (see Figures 6, 7, 8, and 9).

The statechart in Figure 6 mainly specifies the effects of the left and right roll movements. It is considered the highest level of the interaction; first, the user has to indicate at what roller he/she wants to do further. The four main rollers are represented by sep-

arate states in Figure 6: PROGRAM, STYLES, COLLECTION, and RECOMM. Initially, the music style roller has the input focus, i.e., the system is in the STYLES state. By rolling to the right or to the left, one of the other roller will be in focus, i.e., the system is brought in one of the PROGRAM, COLLECTION, or RECOMM states. Additionally, music will be continually played back which will be come more clear in Figure 8 and 9. The music is stopped when entering the STYLES state. For that, there exists some globally defined variable *currTr* which holds the currently selected track. The music will only be stopped by rolling back to the music styles roller. The hierarchical PROGRAM, STYLES, COLLECTION, and RECOMM states are further decomposed in statecharts, as shown in Figures 7, 8, and 9.

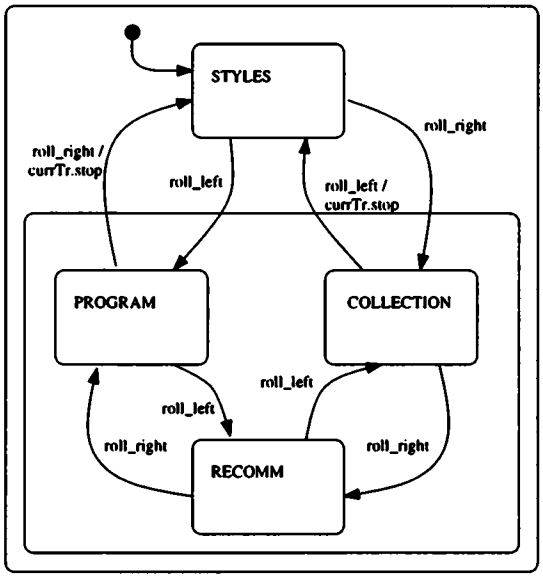

Figure 6. Statechart for roller navigation, i.e., lateral roll movements.

The STYLES statechart in Figure 7 specifies the effects of the roll movements within the music styles roller. The statechart starts off in the CUR_STYLE state. In this state, it is assumed that there exists the notion of a currently selected music style denoted by *currSt*, which is projected at the very front of the roller. Clicking the ball in that state causes some information of the currently selected style to be displayed (e.g., by speech feedback). By a roll on or back movement, the system makes a transition to respectively a NEXT_STYLE or a PREV_STYLE state. If now, *no* so-called roll through movement is made within 300 ms, the system will simply consider the very next or previous style in the list to be the currently selected one. The roller will be visually rotated accordingly. If, however, the roll through movement is made, the system will skip two successive music styles in the list and considers the third one as the currently selected one. This 'power' mechanism to go fast through the music styled explains the need for the roll_on_through and roll_back_through events and their corresponding force fields in Figure 5. Although the 'power' facility is activated by generating two events within a certain short time interval, it is supposed that the user recognizes it as a single hand-stroke that has to be executed more rapidly and to cover a larger distance than the stroke to go along the style one-by-one.

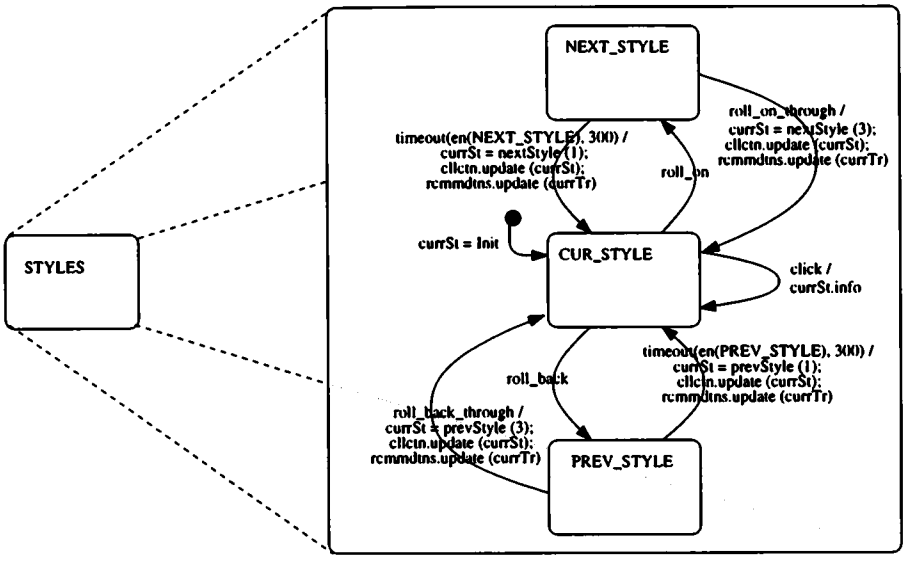

Figure 7. Statechart for rotating the music styles roller.

Going from one music style to another also implies an update of the collection roller and recommendation roller. The tracks that are contained in the newly selected style have to be displayed on the collection roller. The recommendations of the probably newly selection track have to be displayed on the recommendations roller.

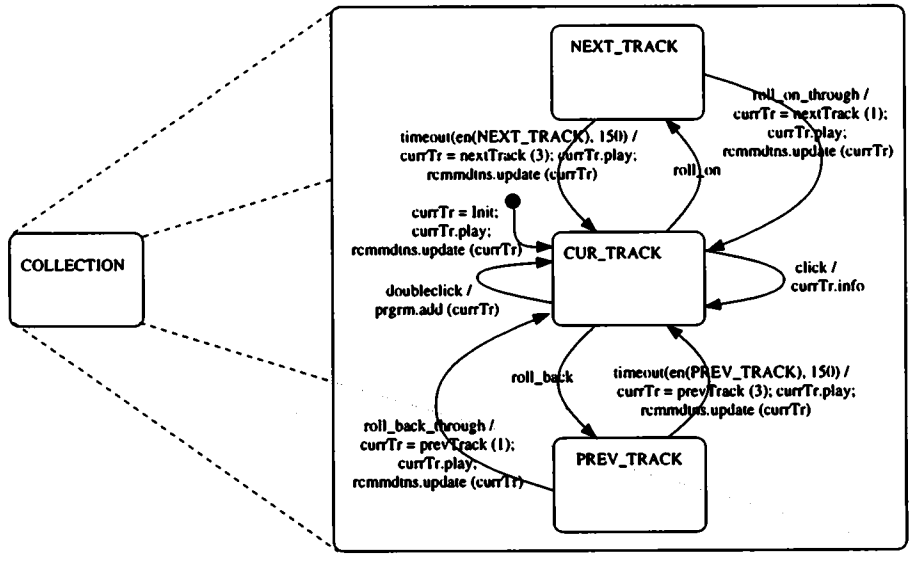

Figure 8. Statechart for rotating the music collection roller.

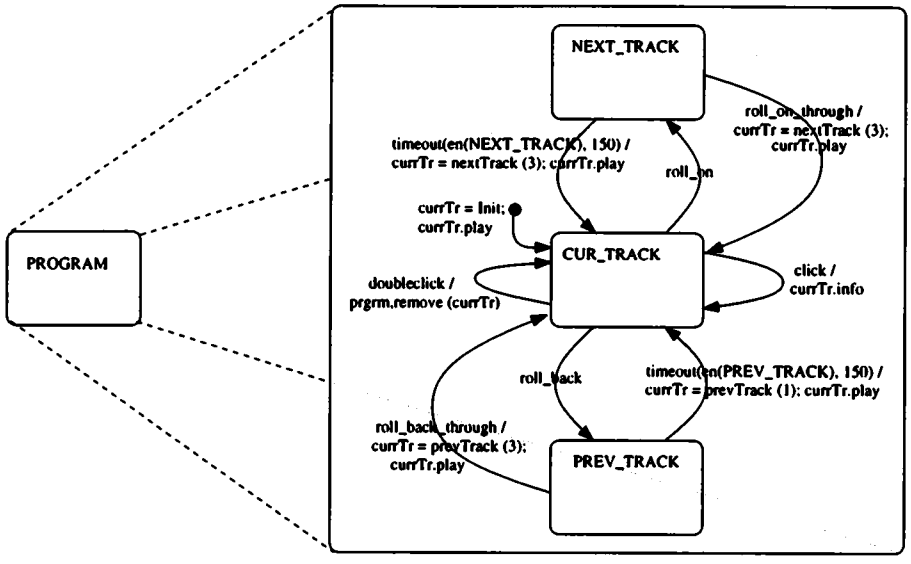

Figure 9. Statechart for rotating the music programme roller.

The COLLECTION statechart in Figure 8 specifies the effects of the roll movements within the collection roller. The same statechart also holds for the recommendations roller, i.e. RECOMM state. The behaviour of the system closely resembles its behaviour in the STYLES case. A currently selected piece of music denoted by *currTr* is assumed to be displayed on the roller, which can be altered by the same roll movements. The currently selected music track is always made audible. A roll movement implies determining the new track, playing back that new track (after stopping the previous one), and displaying the recommendations (*rcmmdtns*) of the new track. A double click of the ball causes the addition of the currently selected piece of music to a globally defined music programme.

The last statechart to be discussed is the PROGRAM statechart as shown in Figure 9. It resembles the behaviour of the COLLECTION and the RECOMM statecharts, with the exception that double clicking the ball now causes the removal of the currently selected piece of music from the music programme. It is assumed here that there are at all times tracks present in a programme.

2.1.3 Multimodal feedback

The statecharts in Figures 6, 7, 8, and 9 do not specify what precise system activities are started, what precise feedback is given, or by what modality this feedback is provided. It only specifies on what pre-defined input events and by what protocol the system will react.

In principle, each system transition from one state to another state is notified by auditory feedback. For instance, going from one roller to the other, or going from one item to another on a roller are signalled by an auditory icon, i.e. non-speech audio (see also Chapter 7). Each roller has its own set of qualitatively different sounding sounds. The music styles roller is featured by sounds that are probably best described as sounds pro-

duced when a metal-like object is hit by another solid object. The music programme roller produces sounds as hitting a glass object by another object. The music collection roller produces wood-like sounds, and the recommendations roller sounds like rubber. Rolling up or down a roller produces sounds from the same quality, but rolling up sounds like a hit further away than a rolling down. The loudness of the sounds are manipulated by adding different noise components, i.e., Poisson or regular pulses. The roll through movements produce sounds of larger objects that are hit, i.e., sounds with a broader spectral bandwidth.

Speech output is used to convey some status information about the interaction. The speech feedback is also given at the system transitions, but with a small latency, to avoid mixing up with other auditory cues. For instance, the system indicates which roller is put into focus, when the ball is rolled to it. In addition, when entering the music programme roller, the system utters how many pieces of music are already added to the programme. The selection of another music style is made known by saying what music style is left for another. At all times, by clicking the ball, the systems utters what music style is currently selected, or to what music style the currently selected piece of music belongs. Speech utterances are interrupted and stopped when their content is no longer valid within the context of the interaction, i.e., when the system transfers to another state.

The following set of utterances is used:

"Programme", "Styles", "Collection", and "Recommendations";

"all styles", and the names of 12 jazz music styles as found in the music collection;

"from ..... to ..... ", in which the slots are filled by names of jazz music styles;

" ..... track", in which the slot is filled by "no" or "one";

" ..... tracks", in which the slot is filled by the numerals "two" up to "twelve", or by the phrase "more than twelve".

2.2 Implementation

The interaction style is implemented in Microsoft Visual Basic 5.0. The manual input and tactual output is controlled by software for the IPO trackball with force feedback and described in Chapter 4 and Chapter 5. For now, it is sufficient to know that force fields have a designated region. When a cursor enters such a region, special events are generated that are made known as mouse move events to the client application. Pushing the ball is passed on as mouse click events. The speech and non-speech audio output are both produced by a component, named ImpactSound, described in Chapter 7. Music is played back by a MPEG Audio player as described in Chapter 8. The general streaming of audio is discussed in Chapter 6. The visualization of the rollers is implemented in the ActiveX control SelectionBall to be discussed in Chapter 9.

As shown in Figure 10, a central role in the implementation is played by the class **OptionBall**, which controls a single roller. It organizes the integration for auditory feedback and visual feedback. For that, it has a private instance to a SelectionBall control (see Chapter 9). In total, three instances of the **OptionBall** class are created, one for each roller containing music, viz., music programme, music collection, and recommenda-

tions. The styles roller is implemented on the basis of another class and the ActiveX control SelectionWheel, but with similar mechanisms. It can be generally stated that the **OptionBall** class is an encapsulation of the statecharts in Figures 8 and 9. The actions to control a roller are notified to the user by audio feedback. For that, it has references to instances of the **Impact** class and the **Utterance** class. Both classes are wrappers to access the ImpactSound DLL; one for playing back non-speech audio, the other for producing speech output. The speech output is based on concatenation of pre-synthesized speech loaded from file. Four resources for speech synthesis were considered, but only the Lucent Text-To-Speech Engine for American English (http://www.bell-labs.com/project/tts/voices.html) is fully implemented. Personal observations and informal discussions with speech researchers concluded that the Lucent speech technology is among the best with respect to the segmental and prosodic quality. Other candidates were

Spengi, the IPO in-house speech engine,

Laureate from BT Laboratories for British English (http://innovate.bt.com/showcase/laureate/index.htm), and

WATSON from AT&T Research Labs for American English (http://www.research.att.com/~mjm/cgi-bin/voices.cgi).

The pieces of music that must be projected by name on the roller are kept in a private instance of the class **MusicCollection**. Obviously, a music collection contains tracks referring to the music pieces.By calling the functions *AddTrack* or *RemoveTrack* this music collection can be assessed.

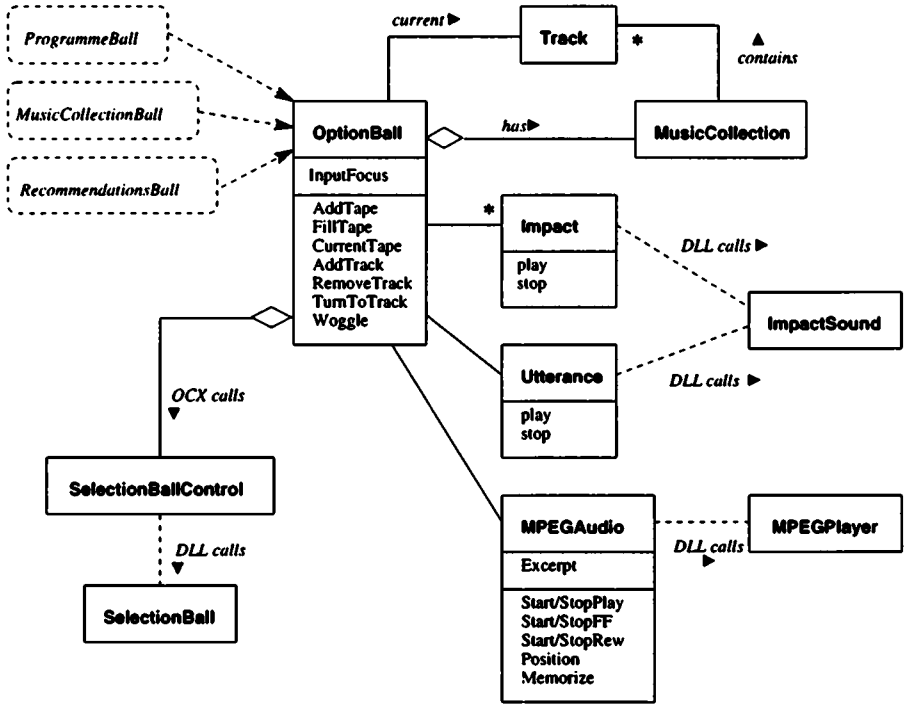

Figure 10. The class **OptionBall** controls the roller and integrates its actions with speech and non-speech audio feedback.

The list of music pieces is wrapped around the roller. The wrapping is done by the SelectionBall component. By calling *AddTape*, a client can add a new empty list to be wrapped around the roller. By calling *FillTape*, this list can be filled with items, i.e., pieces of music or tracks.

An **OptionBall** object has a reference to a track that is *currently* in focus. This current track is displayed at the front of the roller, and its musical content will be played by a global **MPEGAudio** instance. The **MPEGAudio** class is a wrapper of the MPEGPlayer dll.

The attribute data of the music collection is contained in a Microsoft Access database, described in Chapter 3. As shown in Figure 11, the class **MusicBase** provides an easy access to this database. It uses the Microsoft Jet engine and the Data Access Objects (DAO) library for retrieving data from the database. The member function *QueryTracks* retrieves the essential data on music tracks from the database, and sets up an object diagram which mirrors the object model of the database (see Chapter 3). In fact, it builds the whole object structure, as shown in Figure 11, comprising dictionaries containing persons and instruments, and a music collection of music tracks. To let other software entities access this object structure, *QueryTracks* returns a reference to a global **MusicCollection** object, which contains all music tracks in the database. The tracks are one by one added as they appear in the result of the database query. The **MusicCollection** is an extension of the VB class **Collection**. A track is composed of attributes such as title, main artist, and music style. A track has also a list of recommendations, e.g. other tracks, attached to itself. There are, in principle, no restrictions on what tracks are recommendations for others; the composition of the recommendation lists is beyond the scope of this part of the project, but documented elsewhere (Pauws and Eggen, 1996). The recommendations are not read in from the database, but from another resource file.

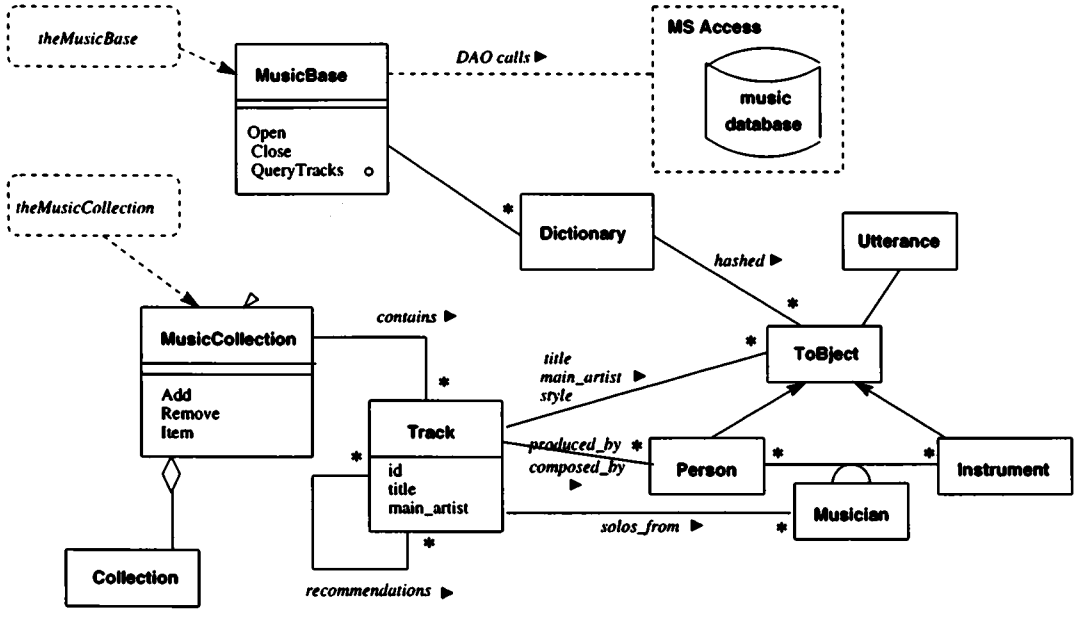

Figure 11. A single call to *QueryTracks* builds an entire object structure representing the essential music database information.

The **MusicBase** object has several instances of the **Dictionary** class, i.e., hash tables, taking care that there is only a single instance of a particular **Person** object, e.g, a person carrying the name 'Miles Davis, or a particular **Instrument** object, e.g., an instrument known as trumpet. The **ToBject** class is used for several things; it can hold 'track titles', 'music style names', or 'names of artists'. A **ToBject** instance can refer to a particular **Utterance** object. **Person** and **Instrument** are both derived from **ToBject.** A **Musician** is a many-to-many relation between a person and an instrument; Miles Davis plays the trumpet, but occasionally keyboards. Trumpets and keyboards are played by many other persons.

3 Music database

A music collection comprising 480 jazz music pieces extracted from 160 commercial CD albums was assembled. A first-minute excerpt from each piece of music was compressed (MPEG 1 Layer II 128 Kbps stereo). The collection was assembled by considering 12 popular jazz genres. These genres cover a considerable part of the whole jazz period. Each genre contained 40 music pieces.

For each piece of music, an elaborate list of attribute data was collected by using information found on the CD booklets and discographies such as 'All Music Guide' (Erlewine, Woodstra, and Bogdanov, 1994) and 'Penguin Guide' (Cook and Morton, 1994). Other data such as tempo, rhythmic structure, and melodic development were acquired by careful listening.

All data of the music collection is put into a Microsoft Access database. An object model of the database is shown in Figure 12. As MS Access is a *relational* database management system, the complete object model is mapped onto tables (Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen, 1991). By mapping objects onto table rows with primary keys, and by mapping the binary one-to-many and many-to-many relations onto distinct tables consisting of a primary key and two foreign keys, the resulting relational database resembles a third normal form. This normal form guarantees a high degree of entity and referential integrity but produces a large number of tables (in this case, 21 tables). A join query is declared which collects and returns the attributes and single-valued relations of all tracks in the database in a single table. Additionally, a select query with a track ID as key is defined for each one-to-many relation.

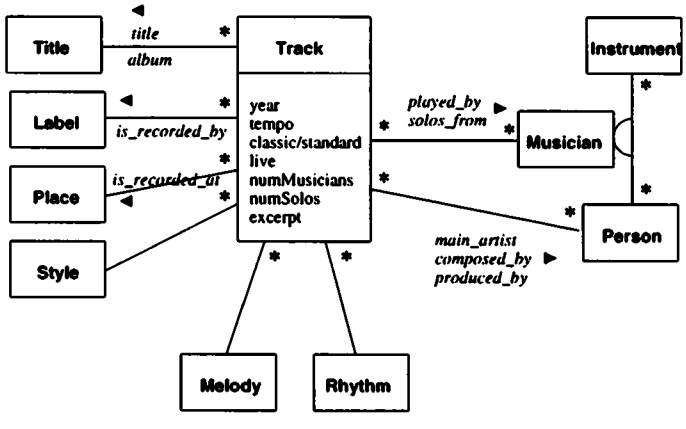

Figure 12. The object model of the music database.

In the object model, as shown in Figure 12, ten entities are defined. As the relations between the entities are quite straightforward, only the entities are further explained.

Track - a recording of a piece of jazz music, with the following attributes

year - year of recording.

tempo - pace of the musical performance measured in beats per minute.

standard - indicates whether the recording is a commonly played tune by jazz musicians that were already popular songs before the heyday of jazz, and thus not primarily intended for jazz musicians.

classic - indicates whether the recording is a commonly played tune composed by a jazz musician or not.

live - indicates whether the recording is recorded in front of a live audience or not.

numMusicians - number of musicians that play along on the recording, ensemble strength (literally: number of instruments played on the recording).

numSolos - number of soloing musicians (literally: number of musicians playing the musical theme of the head chorus in unison or individually).

excerpt - reference to a first-minute excerpt of the recording.

Title - song or album title; both song and album title are included.

Label - the record/distribution company of the recording.

Place - place of recording (e.g. studio, concert hall).

Instrument - an instrument played on a recording.

Person - any person who deserves credits for a recording (e.g., artists, composers, producers).

Musician - a person who plays an instrument on a recording.

Style - the jazz style or era of the recording. The following styles are present:

Blues Jazz comprises rhythm and blues schemes incorporated into a swinging context. Some typical musicians: Billie Holiday, Dinah Washington, Louis Armstrong.

Swing comprises big bands and small groups (1930-1940) that were immensely popular for dance. Some typical musicians: Ben Webster, Count Basie, Oscar Peterson, Toots Thielemans.

Bebop comprises the beginning of modern jazz marked by complex harmonics, up-tempos, and high skills (1940-1950). Some typicial musicians: Charlie Parker, Fats Navarro, Dizzy Gillespie, Stan Getz, Gene Ammons.

Cool Jazz (or West Coast Jazz) comprises the counterpart of bebop and a reaction against its ideas. It is marked by soft melodies and less dynamics. Some typicial musicians: Chet Baker, Gerry Mulligan, Bill Evans, Stan Getz.

Latin Jazz comprises latin rhythms (bossa nova, samba) on which jazz melodies are superimposed. Some typicial musicians: Stan Getz, Jobim, Paquito D'Rivera, Tania Maria, Mario Bauza.

HardBop comprises the return to a more bluesy, less technically demanding melodies than the bebop but with the same intensity (1950-1960). Some typicial musicians: John Coltrane, Miles Davis, Sonny Rollins, Thelonious Monk, Ron Carter, Cannonball Adderley.

PostBop (Modal Jazz) comprises the innovative improvisations on open-ended harmonics rather than strict popular chord schemes (1950-1960). Some typicial musicians: Miles Davis, Herbie Hancock, Wayne Shorter, Dexter Gordon.

Fusion (Jazz-rock) comprises the melting of rock and funk elements with jazz solo techniques. Some typicial musicians: Brecker Brothers, Weather Report, John Scofield, Yellow Jackets, Casiopea.

PostModern (New Age) comprises the influence of world music, classical music, and folk music into the jazz. The composition is far more important than the improvisation. The result can be rather esoteric and atmospheric. Some typicial musicians: Pat Metheny, Paul Bley, John Abercrombie, Bill Frisell, Ralph Towner.

MBase (Avant Fusion) comprises the combination of complex funky rhythms with rather angular melody lines. MBase is short for macro-basic array of structured extemporization. Some typicial musicians: Steve Coleman, Greg Osby, Gary Thomas, Kevin Eubanks, Cassandra Wilson.

NeoBop (Neo classicism, Neo swing) comprises the new generation of young players that find their influence in the acoustic bebop and postbop eras. Some typicial musicians: Branford/Wynton Marsalis, Joshua Redman, David Kikowski, Keith Jarret, Ray Hargrove, Christian McBride.

Dance (DooBop, Jazz-dance, Acid-Jazz) comprises dance-oriented contemporary funk, hip-hop or house. Some typicial musicians: Jazzmatazz, Me'Shell Ndegeocello, Marcus Miller, Miles Davis, Mezzoforte.

Rhythm - rhythmic accompany which characterizes the rhythmic structure and accompaniment made by the rhythmic section of a group. The following categorical values are defined:

3/4 feel - waltz feel.

4/4 feel - swing (walking bass) feel.

5/4 feel - 'Brubeck' feel.

ballad - slow, rubato feel.

latin - traditional latin rhythms such as mambo, samba, and bossa nova.

rock - straight feel, shuffle.

funk - groovy, modern dance-oriented, syncopated feel.

Melody - melodic/harmonic development which characterizes the relation of the chord progression and melody (improvisation) lines of a recording. The following categorical values are defined:

Progressive represents a strong relation between the melody (improvisation) lines and the chords and scales that hold at that moment. The notes are neatly stringed together and cover the whole chordal extension.

Blues represents a melody line (improvisation) linked closely to a former blues chord progression.

Modal represents a *very melodic* melody line on a small chord progression mainly stuck to basic chord notes.

Chromatic represents angular and chromatic melody lines played with some notes outside the chordal context. Angular means large unusual tone interval and chromatic means the opposite: playing notes separated only a single interval.

Non-tonal represents the elimination of western tonality by getting rid of chord scales and key centres. Chords are not played for their resolution but merely for the overall sound.

Free represents the total elimination of chords or other conventions resulting in an intense personal statement of the musician.

4 Trackball with force feedback

The IPO force feedback prototype devices (Engel, Haakma, and van Itegem, 1990) are created in a close cooperation between the IPO and Philips Nat.Lab. in Eindhoven. The one-DOF (degrees of freedom) devices are rotary dials which use either motor- or electromagnetic force feedback. The two-DOF devices are trackball devices with motors attached at the axes of a ball, creating a two-dimensional force plane. Two such devices exist and differ only in minor details. The three-DOF device is a similar trackball device, but, in addition, it has a hinge mechanism with force feedback on which the whole trackball unit can be moved up and down (Keyson, 1996).

Low-cost, smaller devices with a lower power consumption are derived from this technology. Two hand-held prototypes are developed within the programme of the Philips Ease-of-Use Thematic Research for Sound & Vision 1997. These devices have a somewhat limited functionality with respect to the original trackball devices; they are especially designed for usage in a home entertainment environment. One device consists of a small ball that can be rotated fully and freely in one direction. In the other direction, however, the ball can only be moved by a discrete on/off switch. The other device consists of two cylindric freely rotating chambers placed in a perpendicular configuration; one chamber is designated for up/downward rotation, the other for left/right rotation. Because of their small size and hand-held design, they can supplement the current remote controls. Two other low-cost, early commercial versions of the two-DOF trackball are developed within the New Business Creation of Philips Nat.Lab. They are called 'the Peepmouse' and 'the Frog', and occupy only a small region of the desktop. For making 'the Peepmouse', two small motors (from CD-ROM players) were mounted within the housing of a Logitech trackball. 'The Frog' has a housing that betrays the contours of a frog-like creature; the ball is the frog's head. Lied horizontally, this housing facilitates a comfortable rest of the hand, while the fingers are manipulating the ball. Standing upright, 'The Frog' can be used as a game console for video games. For both low-cost versions, similar, but smaller control hardware was used as the original IPO trackball.

The mechanical structure of the original IPO ground-based trackball is patented (Engel, Haakma, and van Itegem, 1990) and will be briefly described here for reasons of completeness. In Figure 13, a representation the trackball is shown; it shows both some of the mechanical as the electronic components of the device. Some components, i.e. the trackball unit, are mounted on a solid plate resting on shock-proof rubbers; the trackball unit can be firmly placed on a desktop. Other components such as the amplifiers and I/O cards comprise another unit. In contrast, on new prototypes such as 'the Frog', the I/O cards are placed within the housing of the device and the power supply for the motors will be tapped directly from the PC. Due to its relatively powerful motors, the original IPO trackball device requires more specialized hardware, i.e., the amplifiers, that had to be placed outside the trackball unit. The trackball unit is covered by a Plexiglass surface on which the wrist can be rested comfortably while the fingers are manipulating the ball.

The trackball is considered a motor input and tactual output device with two degrees of freedom. The hard-plastic ball with a diameter of 57 mm rests on a ball-bearing minimizing any friction. Underneath this bearing system, a contact switch is mounted that

enables the notification of discrete push movements executed vertical to the ball. The apparatus is provided with two independent direct current motors, one for each degree of freedom, i.e., the x and y component. Both rubber-rimmed shaft wheels, each driven by a motor, are fastened at the perpendicular centre lines of the ball. Two small free-rolling support wheels that are fixed at the opposite of each shaft wheel ensure that the ball does not wander from its bearing. The orthogonally positioned shafts enables the creation of a two-dimensional force plane on the ball. Two optical position sensors placed at the end of the motors sense the position of the ball.

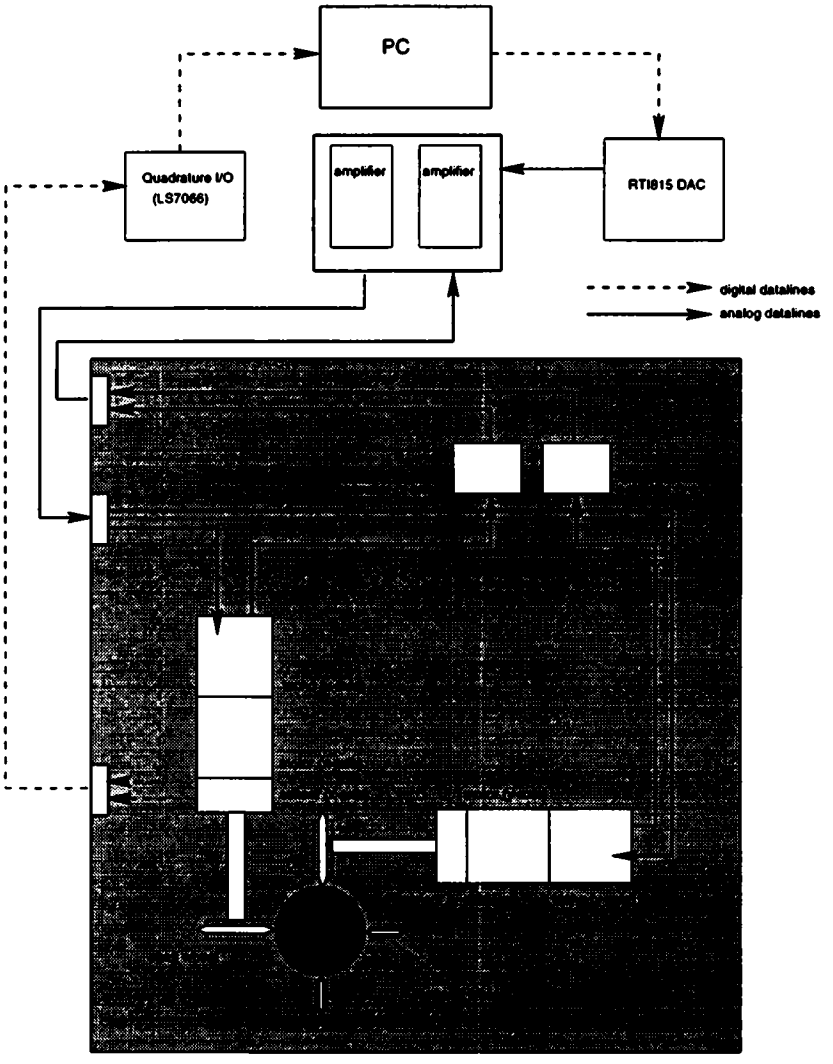

Figure 13. The mechanical and electronic components of the IPO force feedback trackball.

Two independent power amplifiers produce the voltages provided to the motors. These voltages are fixed at a constant level, when the input voltage of the amplifiers is constant. As a consequence, the torque produced by the motor (i.e., the force exerted on the ball) is proportional to the input voltage of the amplifier, but also proportional to the current in the motors. To provide a constant force level on the ball, irrespective of the

ball rotation performed manually, the current in the motor need to be constant. However, a ball rotation that is induced from outside generates an undesired current in the motor. This is compensated by a feedback loop with the amplifier. Voltages corresponding to ball rotation speeds are measured by tachogenerators and voltage dividers and fed back to the amplifier to adjust the voltage provided to the motor, keeping the current level within the motor constant.

The input voltage of the amplifier, on the other hand, is derived from a digital set point value. This set point value is a two-element vector and is interpreted by a Digital-to-Analog Converter (DAC) card, the RTI815. As there is one set point value provided and two motors to communicate with at each time instant, one digital input channel and two analog output channels of the card are used. The RTI815 card can be controlled by software.

One trackball device differs from the other by its motor independent ball positioning sensors. Two optical position sensors, placed on additional wheels, ensure a finer monitoring of the ball position. The optical sensors, whether they are connected on additional wheels or at the end of the motors, are connected to a Quadrature I/O card. The card has a LS7066-chip which can be controlled by software. By means of this software, two data lines (i.e., counters) can be defined representing the x and y component of the ball position.

The hardware configuration of the trackball is controlled by a personal computer.

5 Software for controlling the trackball

The functional requirements of interaction styles have changed due to new demands on integrating more modalities, i.e., peripheral devices, on the input as well as the output side. A current approach in user interface design is an iterative process in which an interaction style can be tested and adjusted 'right-on-the-spot' without an overhead of programming. A rapid prototyping environment is therefore implemented that facilitates construction of an interaction style in which the trackball is incorporated, together with other input and output modalities. If the prototyping environment is in 'design' mode, an interaction style for some application can be interactively developed, mainly by a drag-and-drop philosophy. In run-time mode, the interaction style under construction can be tested. If the interaction style meets its specifications, an executable version of it can be made for the purpose of evaluation, i.e., usage by end-users.

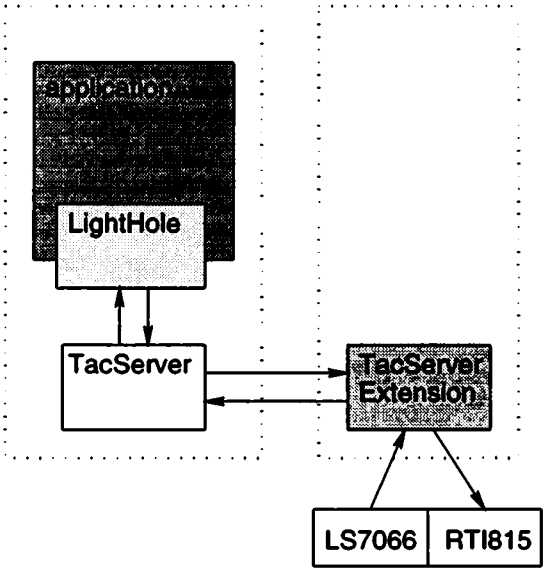

Figure 14. Component-based software architecture of the rapid prototyping environment.

The prototype environment is based on Microsoft technology. The environment currently uses two serially connected PCs; one PC which mainly runs the application and one PC which controls the trackball device. The foreseen required resources for supporting other input and output modalities justify the decision to divide the environment over two PC platforms. The division is not based on conceptual grounds; it can be made undone without much effort.

A component-based design strategy is used, which facilitates easy change and exchange of software, e.g., when other force feedback devices have to be connected to the environment. Three in-house software components are developed that make up the rapid prototype environment, as shown in Figure 14. They were deliverables of the Philips Ease-of-Use Thematic Research for Sound and Vision Programme 1997.

- LightHole, an ActiveX control component that can be directly incorporated in an application without any hard-coding. This component facilitates the specification where, when, and how force feedback is evoked in the application.
- TacServer, a static link library, which facilitates the communication between the two PCs. Therefore, it communicates asynchronously (by a RS-232 connection) with the PC that actually controls the trackball device.
- TacServer Extension, a program that runs on the separate PC dedicated to control the trackball device by its RTI815 and LS7066 interfaces.

Before the components are described in more detail, the underlying concepts that abstract from the low-level control of the trackball device are outlined.

5.1 Concepts

The addition of force feedback in an interaction style is based on the concept of laying out tactual objects, i.e. TouchCons, in a workspace. Some of these objects evoke a local force field when their designated region is passed through by a cursor. As such, these objects have a certain definite region of action. Others call forth a global force field when other conditions hold, such as timing conditions. Workspaces consist of a collection of tactual objects that supply a certain functional coherence, but they can be interpreted in various ways by UI designers. For instance, a workspace might refer to certain physical regions on a visual display; a workspace maintains then a one-to-one correspondence with the pixel coordinates on the visual display. But workspaces, i.e., Visual Basic forms, might also refer to well-defined parts of a dialogue that can be called up during run-time.

The control of the cursor is mediated by the trackball, but it can, in principle, be controlled by any other input device. Although the tactual objects are represented by graphical objects, the tactual objects do not to be visualized in the interaction style. Also, the cursor may not be rendered on the display. The force feedback is mediated by a so-called haptic output device, such as a force feedback trackball or force feedback joystick. In the case of the trackball, the force is passed through as a system-driven ball movement in a specific direction.

Tactual object names such as 'hill', 'peak', 'hole', 'path', or 'bump' refer to real-world objects that have a direct mental imagery, but they also have an association to their tactual sensations. This iconic representation of tactual objects facilitates easy recall and a common language among designers (Keyson and Tang, 1995; Keyson, 1996; Keyson and van Stuivenberg, 1997). When rolling the ball, a 'hill' feels like moving up some inclination, until a turning point is reached. After this point, ball rolling is at best described as 'running down the hill'. Whereas 'hills' have a convex inclination described by a curve, 'peaks' have a far more pronounced force field described by straight lines.

Technically, each tactual object represents a parameterized two-dimensional tactual force field map. The map is described by geometrical formulas. The force field map of a 'hill' evokes a directional pulling-force away from its force field centre, when the cursor is moved towards this centre. Definitions of some force field maps are described else-

where (Keyson and Tang, 1995; Keyson, 1996; Keyson and van Stuivenberg, 1997). Tactual objects can be grouped to design more complex, but re-usable force field patterns.

5.2 Lighthole

The ActiveX control component LightHole enables a flexible integration of the tactual objects in an application. Each instance of the ActiveX control represents a tactual object. The controls themselves are visually represented by graphical draw primitives such as circles and rectangles; their graphical representations can be hidden when the interaction style actually runs. The controls can be declared by simply selecting and placing them on the application's workspace, i.e., Visual Basic forms. The control's parameters are editable by means of dialogue boxes, i.e., property pages, and correspond to parameters of a tactual object. Obligatory parameters are the name of the declared object (control), the position, size, and orientation of its force field, and the maximum pulling-force.

A set of 12 tactual objects is currently implemented (Keyson and Tang, 1995; Keyson, 1996; Keyson and van Stuivenberg, 1997), but new objects can be integrated with little effort, as will be illustrated in Section 5.4.1. Though it requires programming at all three components, the software utilizes implementation inheritance which support easy integration of new tactual objects.

When the application runs, the LightHole component calls the TacServer library for actually delegating the creation of the tactual objects and the corresponding force field maps. After that, it handles and processes window events such as cursor position changes, and button press events, in an ordinary window event processing function. As will be described in Section 5.3, the relevant window events actually originate from events of the trackball device. For instance, cursor positions (i.e. mouse movement events) correspond to positions of the trackball.

5.3 TacServer

The static link library TacServer is a 'service hatch' between the LightHole component and the TacServer Extension; it serves as a bridge between two software component which can vary now independently. An asynchronous communication protocol is specified between the TacServer and its Extension, in which the creation of tactual objects, their parameters, their state, but also ball push events, and ball rotation speeds are exchanged by packets (byte sequences).

At both sides, the communication protocol is encapsulated by **commLink** objects. Clients are allowed to invoke its member function *sendPacket* for sending a packet of a specified size, or *peekPacket* for peeking a packet with a specified packet identifier. The size of packet is not fixed, but depends on the content of the message. A packet that communicates the latest cursor or ball position occupies only 5 bytes; a one-byte packet identifier plus four bytes for transmitting the x and y coordinates. However, a packet that communicates the request for a creation of a tactual object has a bigger size; its size depends on the number of parameters to be transmitted. For instance, a 'hill' object is specified by a relatively small number of parameters. It requires a 15-byte packet for

communication; a one-byte packet identifier that announces that the packet deals with an object creation, one byte that specifies what object needs to be created, four bytes that specify the centre of the hole, 8 bytes that specify the four radii of the hill's elliptic region of action, and one byte that specifies the maximum pulling-force at the centre of the hole. The size of a packet is not explicitly encoded in the packet by client objects, but is accounted for in the communication protocol. However, packets are not allowed to exceed 256 bytes. At the recipient site, a maximum of 32 packets (32*256 = 8K) are allowed to pent for further action of the recipient, otherwise a communication overflow occurs.

TacServer contains an instance of the class **Tpointer** which has for each piece of information, that needs to be exchanged, a designated member function. For instance, there are member functions for creating a tactual object at the recipient site, or a member function that asks for a peek at the control hardware of the trackball device, or (re)setting the absolute position of the ball. Each such member function can be invoked after the communication link is set up by instantiating a **commLink** object. The member functions consist mainly of code to package the information and sending it off to the TacServer Extension.

In addition, TacServer continually request the TacServer Extension to peek the control hardware of the trackball device. To notify whether there are any changes with respect to ball- or button presses, or to ball positions or speed, it peeks to received packets that might contain such information. Any change in status information is immediately coerced to MS-Windows events. Subsequently, these MS-Windows events can be interpreted by the LightHole component in its window event processing function. For instance, ball positions coordinates are directly translated to cursor positions and ball pushes simply correspond to mouse clicks.

5.4 TacServer Extension

The TacServer Extension is based on previous versions of a rapid prototyping environment for IPO trackball devices called TacTool which was developed at the IPO (Keyson and Tang, 1995; Keyson and van Stuivenberg, 1997). The TacServer Extension is an implementation of the tactual object concept to control the trackball device. As already mentioned in Section 5.1, tactual objects are placed in so-called workspaces; this also applies in a software-technical way. Only a single workspace is active at the time. When active, a workspace receives all input events and all incoming communication packets. However, another workspace can be made active by a switching mechanism. By this way of working, workspaces have the potential to extend the physical proportions of a display, creating some 'virtual space' beyond the visual display. Currently, for each Visual Basic form, LightHole creates a separate workspace. If new tactual objects are put onto a form, it is checked whether this form is already associated with a workspace by looking at its window handle. If not so, a new workspace is created. If an application has placed tactual objects on a single form, only the default workspace is created.

User input events originate, in general, from the control hardware of the trackball device. By peeking this hardware continually, one is notified by ball presses or changing ball positions. These events, besides being communicated to the TacServer, are dispatched to the current active workspace. Subsequently, the events are directed to all tac-

tual objects within this workspace in a list order, i.e., in the same order as the tactual objects are added to the workspace.

If the current ball position is within the tactual object's region of action, the tactual object acts on that by calculating a force vector. The value of the force vector is determined by the force field map of the tactual object. When all tactual objects have provided their force vectors, all force vectors are summed to arrive at a global force vector. This vector is send off as the digital set point to the RTI815 card.

Visiting all tactual objects one-by-one implies that the dispatching of user input events is dependent on the order in which tactual objects are created and added to the workspace. However, sending off events this way has no undesired side-effects, because only an overall force vector is calculated. User input events such as a ball-pushes are sent off to the TacServer and the LightHole components. As the force calculation and the handling of other user input events (and the execution of the application) take place at different PC platforms, synchronization facilities are built in that compensate for any misalignments.

5.4.1 Implementation

The first thing that meets the eye when one looks more closely to the source code is the profound use of inheritance for implementing all kind of list types. These list types are all derived from a basic list type. They are specialized based on their constituent objects; for each object class that must be contained in a list, there is a designated list class. In this way, the list classes fulfil the role of wrappers that coerce undefined types of list elements to a well-defined type. Although this design pattern surely encapsulates type-casting and type-coercion, it results into a considerable number of exotic list classes (the 'ravioli effect').

5.4.1.1 Controlling the trackball device

TacServer Extension is built around an instance of class **TsmartBall** which incorporates all functional essentials as shown in Figure 15. It maintains all workspaces from which only one is active. Additionally, it contains two instances of a communication class **TcommLink**; one for sending packets, and one for receiving packets.

The member function *run* is the core process of TacServer Extension. It is mainly an endless loop, waiting for flags to be put on, and subsequently react on these signals. Several timers interrupt at a pre-defined time interval and invoke an interrupt handler that set those flags. There are flags (and corresponding timers) for reading the ball position, determining the ball speed, do some communication and so further. As an example, the ball position is updated (i.e., read) 400 times per second. Immediately following a ball position update, the force field in the current workspace is adjusted. If required, packet communication can be done 25 times per second. Dependent on the received packets, the TacServer Extension can perform the following actions:

- new tactual objects can be created and installed in the current workspace;
- tactual objects can be deleted and removed from the current workspace;

- new workspaces can be created and added to the collection of workspaces;
- workspaces can be deleted and removed from the collection of workspaces;
- a switch from one workspace to another workspace can be made;
- the absolute position of the cursor can be set in the workspace;
- a synchronization measure between TacServer and its Extension.

In addition, the current ball (cursor) position is shared with the TacServer by packet communication.

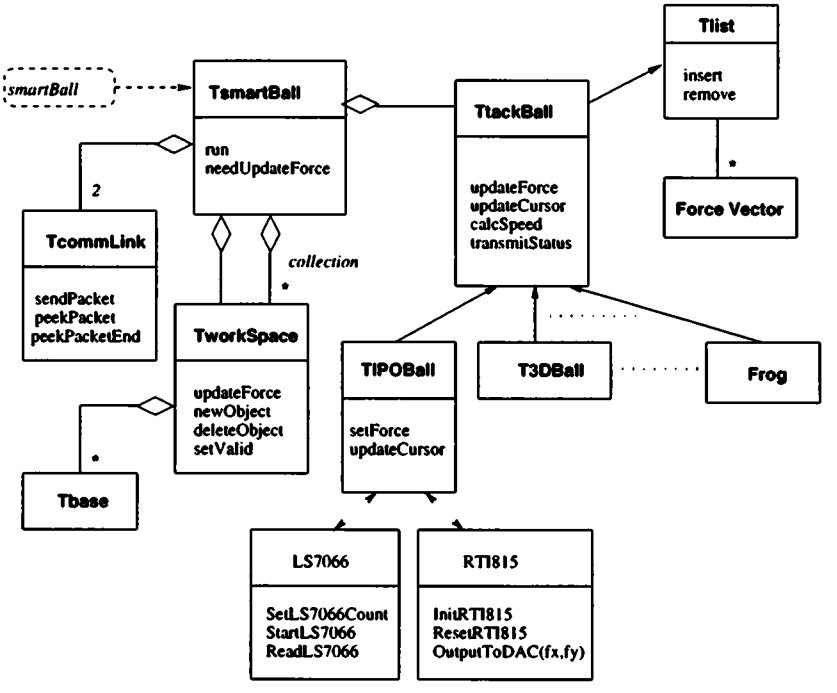

Figure 15. The class diagram of TacServer Extension that incorporates most functionality.

Functionality for controlling the trackball hardware is declared in the abstract class **TtackBall**, also shown in Figure 15. It essentially abstracts an interface for specific control hardware, i.e., other trackball devices. This makes TacServer Extension independent of the trackball hardware peripherals. Subclasses of **TtackBall** are specialized in specific trackball devices such as the original IPO ball, i.e., **TIPOBall**, or its 3D equivalent, i.e., **T3DBall**. Other devices such as 'the Frog' or 'the Peepmouse' can be easily integrated. A single instance of one of these subclasses is created at run-time.

The **TtackBall** class is essentially a list; it contains references to force vectors currently emitted by tactual objects. The **Tlist** class embodies an complete implementation of a linked list data structure, which can hold references to objects of an undefined type. The member function *updateForce* computes the sum of all inserted force vectors, and gives this value to the DAC component RTI815 by the member function *setForce*. The member function *updateCursor* calls on the LS7066 for retrieving the current ball position. The

two most recent ball position are used by *calcSpeed* to estimate the current rotation speed of the ball. The current status of the ball, i.e. its x, y - position, is transmitted by *transmitStatus*. It needs therefore a reference to a communication object **TcommLink** in its argument list.

5.4.1.2 Workspaces, tactual objects

As already mentioned in Section 5.1, workspaces are the coherent building blocks of the interaction style. They might refer to a portion of the screen, in which the cursor is moved. As shown in Figure 15, a workspace holds a list containing all kind of tactual objects (derived from **Tbase**) which together set up a global force field in the workspace. Although we only refer to tactual objects, there are also other entities that can be added to a workspace. As shown in Figure 16, the complete lists is as follows.

- Tactual objects (derived from **Ttactilebase**) set up a passive, local force field dependent on the current ball position and a force field map description. Examples are 'hole', 'bump', or 'hill'.
- Effect objects (derived from **TeffectBase**) set up a active, merely a global force field that is applied at some time interval. The effect is described by a force field map. Some effects objects are dependent on both current ball position and time. Examples are 'conveyer belt' and 'swirl'.
- Clipper objects (derived from **TclipBase**) set up a geometric clipping region. They disallow regions to be reached by the trackball, by resetting the absolute position each time a clipping region is entered. One can clip by line (also known as 'the wall'), circle, arc, or box.
- Trigger objects (derived from **TtriggerBase**) set up regions that notify the TacServer upon entering or leaving. These objects are superfluous, as the events of entering or leaving are handled by TacServer and LightHole.
- Group objects (**TgroupBase**) gather a set of objects and treat them as a unit.

Each tactual object class (e.g., **TtactileHole**) contain private data that specifies its force field map and a private 2-element force vector. The force vector is calculated dependent on the current x, y ball position and the force field map by the member function *forceAt*. Upon calling *updateForce* in the workspace, the task is delegated to all tactual objects added to the workspace. One-by-one, it is determined whether the current ball position is within the reach of a tactual object (by means of *hitTest*) and the force vector is updated. If the ball enters, so-to-say, a tactual object, a reference to the object's force vector is inserted to the list of force vectors maintained by the **Ttackball** class (recall **Ttackball** is a list structure as shown in Figure 15). If the ball leaves the region of a tactual object, the reference to the object's force vector is removed.

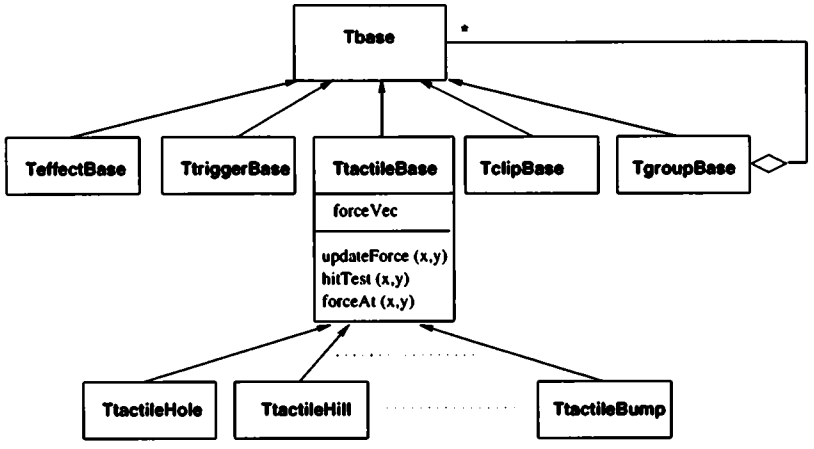

Figure 16. Several distinct tactual objects define a force field map that is mediated through the trackball device.

5.4.1.3 Creation of tactual objects

Because TacServer Extension is responsible for creating tactual objects on-the-fly, dependent on communication packets received from the TacServer, it can not predict what tactual subclasses to instantiate. It knows only *when* to create an tactual object, and to *what* workspace to add it, but needs to decode *what kind* of tactual object to create. Therefore, the knowledge of which tactual object to create is encapsulated in a well-known design pattern, the Factory Method (Gamma, Helm, Johnson, and Vlissides, 1994). In this pattern, tactual objects are created by specific builder objects, such as shown in Figure 17 for the **TtactileHole** class. An abstract class **TbaseBuilder** declares the *build* operator, which returns an object of type **Tbase**. But once a subclass of **TbaseBuilder** is instantiated, it instantiate the appropriate tactual objects without actually knowing their class by overriding this abstract *build* operation. The *build* operation receives the communication packets in its argument list; the creation of an tactual object relies on unpacking this packet for its content (e.g., parameters of a force field map) and directly calling the constructor of the tactual object class.

Also shown at the top of Figure 17, all builder objects are contained in a for-that-purpose-appointed collection, i.e., **TbaseBuilderCollection**. This collection is a type-coercion wrapper of a sorted list class which facilitates a binary search method for finding list elements. The appropriate builder object is now looked up in the collection by means of a tactual object identifier, i.e., OBJ_ID. The object identifier OBJ_ID is encoded in the communication packet.

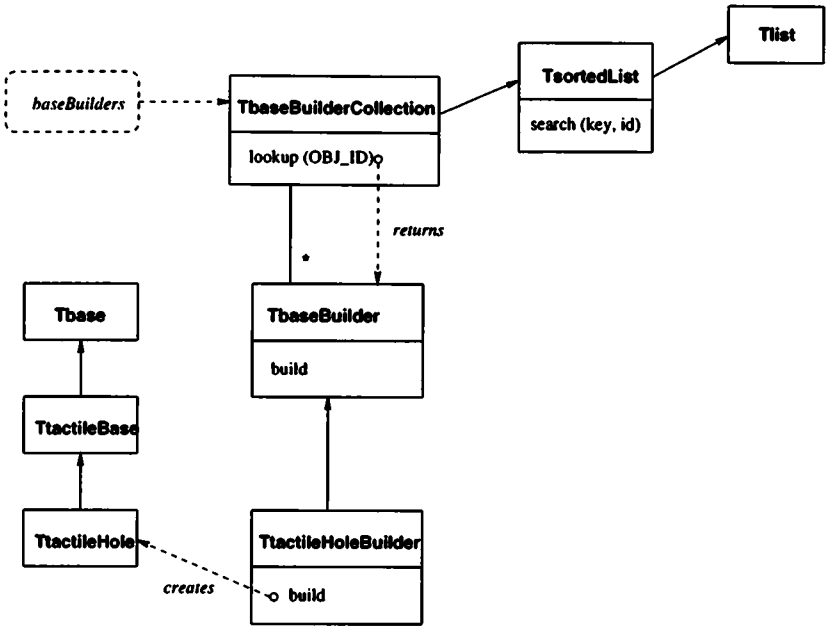

Figure 17. A concrete builder object redefines the abstract operation of a build operation by returning an appropriate tactual object.

Obviously, the functionality of the design pattern as shown in Figure 17 could also be implemented by an ordinary `case` statement structure, but this does not support easy extension and reusability of software. Now, new classes of tactual objects can be added to the TacServer Extension with little effort. It only needs adding the implementation of a subclass to the class diagram as shown in Figure 16, writing its builder class as shown in Figure 17, and slightly adapting the communication protocol between the TacServer and TacServer Extension. Obviously, code for creating tactual objects in a client application must also be added in the LightHole component.

6 Software for streaming audio output

As the requirements on control, speed, and the diversity of input/output modalities are increasing for interaction styles, one must rely on software packages that provide features that harmonize some of these requirements. The audio output requirements of this project's interaction style were rather harsh. Audio data streaming from distinct sources at once, i.e., high-quality music audio, synthesized speech, and non-speech audio, had to be mixed and made audible with low-latency and with a maximum control to intervene the streaming. Streaming audio output is therefore implemented by using DirectSound 5.0 object technology, one of the components of Microsoft's DirectX package.

The DirectX package is the Windows' multimedia application programming interface (API) for developing graphics, animation, rendering, sound, input, force feedback, and multi-player network games. The near future promises that this API will be supported on any Windows platform. It is marketed as a product that provides almost asynchronous low-level access to multi media hardware in a device independent way without delay latency. All DirectX object classes provide a functional set of methods on a variety of multi media platforms, which interfaces are compliant to the Component Object Model (COM). If the underlying multi media hardware has the appropriate capabilities, the implementation of these methods will *directly* call on the hardware known as hardware acceleration. Otherwise, the methods will be emulated in software and system memory (Bargen and Donnelly, 1998).

DirectSound functionality is built round the concepts of a *primary sound buffer* and a virtually unlimited number of *secondary sound buffers*. Each secondary sound buffer contains a single sound in the form of pulse code modulation (PCM) samples. The buffer may contain a static sound or streaming sound. A static sound is put there once, stays there for a while, and will be played back repetitively. Streaming sound, on the other hand, needs to be transferred block by block into the buffer while playing simultaneously; it may be the result of another process, e.g., an audio decoding process, or the data block is just too big to be copied at once. When a audio play back function is issued on a secondary sound buffer, its content will be mixed with all other secondary sound buffers that are currently playing. The resulting mix will be put into the primary sound buffer and is directly sent off to the audio output device. Besides some privileged audio output format setting, clients do not have access to the primary sound buffer. In general, they are only allowed to fill or read from the secondary sound buffers.

Although DirectSound is used for this project to facilitate streaming audio output, it is not yet a standard used by all software components from third parties such as speech recognition engines. These components have their own *hidden* strategy for capturing speech from an audio input device, e.g., by using the Win32 waveform API. There is no direct way to intercept these calls from speech recognition components, or to synchronize it with calls to DirectSound. Up until now, any hardware conflict that arises from the integration of software components that use different strategies to capture or write audio must be resolved, literally avoided, by inserting an additional audio card. Another remedy is to alternating requests to services of DirectSound and calls to third-party software.

6.1 AudioDevice

Because DirectSound is loaded with a extensive set of features and requires a disciplined and defensive style of programming, an object class called **AudioDevice** is wrapped around the DirectSound functionality. It provides a coherent set of functions that abstracts from the DirectSound interface. Additionally, it takes away the responsibility of defensive programming, i.e. continually checking for a possible error status, from the client.

As shown in Figure 18, the class **AudioDevice** has transformed the concept of a secondary sound buffer into the concept of an audio port. An audio port class called **AudioPortInfo** groups all variables and structures that are required to control a single secondary sound buffer. It keeps a private instance to a **WAVEFORMATEX** structure, which contain format information on the PCM data in the buffer (e.g., sample frequency, precision, mono/stereo). It also has a private instance of both a secondary sound write buffer (i.e., **LPDIRECTSOUNDBUFFER**) and read buffer (i.e., **LPDIRECTSOUNDCAPTUREBUFFER**). Associated with each write and read buffer, there are two instances of respectively **LPBUFFERDEC** and **LPCBUFFERDESC**. These classes provide control parameters such as volume level and panning for processing the buffers accordingly.

In fact, DirectSound 5.0 also provides an API to read, i.e., capture data from an audio input device. Although the **AudioDevice** class works along similar mechanism for both playing and reading audio, its DirectSound implementation currently lacks hardware acceleration under Window 95. As it is now, the DirectSound API for capturing data is a COM wrapper for the Win32 waveform API. Direct hardware access will be present in Windows 98 and NT 5.0 (Bargen and Donnelly, 1998).

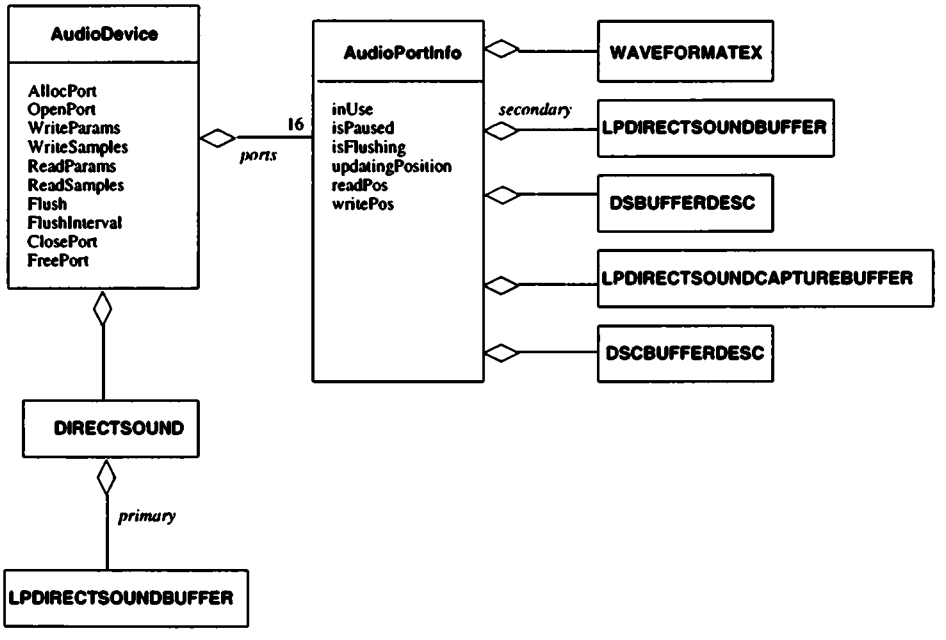

Figure 18. The class diagram of the AudioDevice.

Upon creation, **AudioDevice** initializes a **DIRECTSOUND** object, which mainly manages the primary sound buffer. Because only a single **DIRECTSOUND** object is preferred to persist during the life-span of an client application, there is also a single instance of **AudioDevice**; the **AudioDevice** is a (static) singleton class. It is important to know that DirectSound has defined four distinct cooperative levels that determine how the audio output device is shared among multiple client applications, i.e., windows (Bargen and Donnelly, 1998). **AudioDevice** has itself privileged to change the audio format of the primary sound buffer, thus to override the default setting. This may also cause changing the output format of other applications. If a more cooperative level is required in which the output format is not that easily altered by a client, necessary changes to the code of AudioDevice has to be made. In addition, DirectSound requires a direct binding with a window; it can serve only one client at a time. By default, **AudioDevice** binds DirectSound to the foreground window. This binding can be overruled by a client application by passing another window handle.

The **AudioDevice** class holds references to 16 audio ports; each port will be allocated upon explicit request of the client application. The presence of a maximum of 16 audio ports results into an equally number of secondary sound buffers. Thus, at maximum, 16 different sounds can be mixed together and made audible simultaneously.

The class **AudioDevice** only supports streaming audio. When data is provided to the instance of **AudioDevice**, it is immediately played back; there is no facility that keeps audio data permanently. The implementation of streaming audio within **AudioDevice** is done by the DirectSound concept of *circular* secondary sound buffers. New blocks of data need to be timely provided at the right place, before the mixing procedure, going along circular buffers, revisits this place. Therefore, DirectSound has introduced concepts like a 'current play position' and a 'current write position' in the buffers, which respectively point out what region of the buffer is safe to write data to, and what region is currently used for mixing. To investigate whether it is possible to place the next portion of data, **AudioDevice** continually polls the current state of the play and write positions in a separate thread[1]. Instead of polling, one could also decide to use a notification scheme or timer interrupts, but polling costs the least programming effort. Although **AudioDevice** keeps track where and in what order to put the next block of data, it is still the client's responsibility to pass a timely next block.

A feature which is not explicitly available in the DirectSound API is a flush facility. A flush enforces pending audio data in the secondary sound buffers to be sent off to the audio output device. Flushing is needed when all audio data is streamed to the **AudioDevice** but is not yet processed, before audio ports can be freed to be used by other clients, i.e. threads. The function *Flush* is implemented by a busy wait mechanism and does not return until all data is processed. The function *FlushInterval*, on the other hand, only returns whether there is still some data that is currently processed; it provides a finer control in multi-threaded environments.

---

1. Threads differ from conventional processes that they do not have a separate address or code space. A thread shares code and address space with other threads and, because of that property, is sometimes called a lightweight process. A thread does have an individual program counter and a stack for local variables and arguments. Within software engineering, the names 'task' and 'job' are reserved for referring to the static purpose that processes and threads have to fulfil; threads and processes merely refer to run-time behaviour of software.

One of the imperfections of DirectSound 5.0 is its lack of an adequate sample rate conversion. Whenever audio data in the secondary sound buffers comply to different audio formats, DirectSound carries out a crude up- or downsampling technique to arrive at an uniform audio format. This leads to an unacceptable audio quality. It is therefore recommended to perform an explicit high-quality sample rate conversion to an uniform format before audio data is streamed to the **AudioDevice**.

6.1.1 Interface at client application level

The AudioDevice is compiled and linked into a Dynamic Link Library (DLL) which can be used in C or C++ applications. The most important portion of the DLL interface is as follows.

```
int AudioDevice_Reset ( HWND *hWnd );

long AudioDevice_AllocPort();
int AudioDevice_OpenPort( long portnum, char *mode );
int AudioDevice_ClosePort( long portnum );
int AudioDevice_FreePort( long portnum );
int AudioDevice_IsPortOpen( long portnum );

int AudioDevice_WriteParams( long portnum, char *mode,
        long numChannels, long sampleSize, long sampleRate);
int AudioDevice_WriteSamples( long portnum, void *buf,
        long numSamples, int bLastOutput );
int AudioDevice_StopWriteSamples( long portnum );

int AudioDevice_ReadParams( long portnum, long *numChannels,
        long *sampleSize, long *sampleRate );
int AudioDevice_ReadSamples( long portnum, void *buf,
        long numSamples );

int AudioDevice_Flush( long portnum );
int AudioDevice_FlushInterval ( long portnum );
int AudioDevice_Play( char *fileName, int wait );
```

All functions, except for `AudioDevice_AllocPort`, return an integer value indicating whether errors have occurred while calling DirectSound. Because the number of error types are quite numerous, **AudioDevice** intercepts these errors and sets the return value to 1. Additionally, it pops up a small window in which the error description can be read off. If the functions return 0, all went well.

A client might pass a window handle to the device by calling `AudioDevice_Reset`. Consequently, the device dedicates all its resources to that window. As a side effect, all audio ports are freed, i.e. de-allocated, from the current window.

Before any audio data can be streamed, an audio port has to be explicitly allocated. The function `AudioDevice_AllocPort` returns an indentifier to an available audio port, which can be used in subsequent calls to the device. If this function returns a port number that is less than zero, the audio device was not capable of allocating an audio port. Probably, all ports (from 0 up until 15) were already occupied. Subsequently, the audio format for the audio port has to be specified by invoking `AudioDevice_WriteParams`. Clients can query the audio format of an audio port by calling `AudioDevice_ReadParams`. The audio format is specified by the intended mode of usage (`"w"`: writing or `"r"`: reading), the number of channels (`1`: mono, `2`:

stereo), the sample size (i.e., precision in number of bits: 8 or 16), and the sample rate measured in Hertz (e.g. 8000, 16000, 32000, 44100). Then, the audio port needs to be opened by indicating, again, its intended mode of use. After this initialization, the client is free to stream audio data to the device, or read audio data from the device by using respectively `AudioDevice_WriteSamples` or `AudioDevice_ReadSamples`. These functions are asynchronous, i.e. they return immediately. Both functions need a pointer (`buf`) to an allocated block of short integers, and the size of this block (`numSamples`). Blocks in stereo format organizes left-channel and right-channel data points in alternating order per sample. When writing samples, the client is also obliged to inform the device when the current block of data is the last block to be transferred by setting the boolean `bLastOutput` accordingly. Playing back audio data at an audio port can be immediately stopped at all times by invoking `AudioDevice_StopWriteSamples`. When all data is streamed, one can flush the device, if necessary, before closing and freeing the audio port.

The additional function `AudioDevice_Play` circumvents above strategy. It can be used when a client only wants to play back audio from an audio file in WAV-format. The boolean `wait` indicates whether this call must be asynchronous, i.e., starting up a separate thread, or not. However, the effect of an asynchronous call can not be made undone; playing back audio can not be interrupted.

6.1.2 Source code example

A C++ code excerpt for playing a 2-second 500 Hz sine wave sampled at 8 kHz, 16-bit precision is as follows:

```
#include <math.h>
#include "adev.h" /* header file for AudioDevice */
#define M_PI 3.1415926535

int main( int argc, char **argv ) {
       int status = 0;
       long pnum = -1;

       pnum = AudioDevice_AllocPort();
       if ( pnum < 0 ) status = -1;
       if ( status == 0 )
          status = AudioDevice_WriteParams( pnum, "w", 1, 16, 8000 );

       if ( status == 0 )
          status = AudioDevice_OpenPort( pnum, "w" );

       if ( status == 0 ) {
          short signal[16000];
          int i;

          for( i = 0; i < 16000; i++ )
                signal[i] = (short) (10000.0 * sin( 2.0 * M_PI * i / 16 ));

          status = AudioDevice_WriteSamples( pnum, signal, 16000, 1 );
   }
       if ( status == 0 )
          status = AudioDevice_Flush( pnum );

       if ( status == 0 )
          status = AudioDevice_ClosePort( pnum );

       if ( pnum >= 0 )
          AudioDevice_FreePort( pnum );
}
```

7 Software for generating non-speech sound

In recent years, the unclear role of non-speech sound and the requirements of hardware and software resources have delayed the use of sound in interaction styles. However, interpreting sound in a framework of *everyday listening* (Buxton, Gaver, and Bly, 1992; Gaver, 1997), in which sound percepts are directly mapped to source attributes and real-world events, seems to justify an appropriate use of sound, i.e. auditory icons, in an interaction style. When sound is synthesized rather than pre-recorded and sampled, it provides loss of storage requirements and a larger flexibility in sound design. When sound is, in addition, synthesized by acknowledging sound sources and real-world events, it gives an opportunity to instantiate theories on real-world sound perception.

The non-speech sounds, used in this project, are generated by means of a Constrained Additive Synthesis technique (Buxton, Gaver, and Bly, 1992). This technique is based on the addition of elementary signal components (or partials) such as sinus waveforms (pure tones), block (square) waveforms, sawtooth waveforms, or triangle waveforms. These components are described by parameters such as centre frequency, amplitude, duration, or phase shift. According to Fourier theory, any complex sound can be decomposed into an indefinitely long list of sinus waveforms. Obviously, this also holds for the signal components used in the synthesis technique. For instance, the rather harsh sounding block waveform consists of a fundamental sinus waveform with the same centre frequency and all its uneven harmonics with amplitudes in the ratio of $1/n$. On the other hand, the flute-like triangle waveform is made out of the same components as the block waveform, but the amplitudes of the harmonics are in the ratio of $1/n^2$. Another harsh sounding waveform, the sawtooth waveform, consists of all harmonics of a fundamental sinus waveform with amplitudes proportional to $1/n$. It is thus sufficient, in principle, to refer only to sinus components.

Each component has also an amplitude envelope which describes the amplitude onset and amplitude decay characteristics. The envelopes are described by exponential or gamma curves. Addition of these amplitude-adjusted components yields a finite signal form which can be excited by a pulse train. Convolution with a pulse train (e.g., single pulse, regular pulse train, Poisson pulse train) means, besides a amplification (or attenuation) factor, the positioning of multiple, possibly overlapping, signal forms in time.

Techniques, other than the Constrained Additive Synthesis, e.g., the Constrained Subtractive Synthesis, are based on the application of filterbanks. Our applied technique is however considered especially efficient for the generation of impact sounds; the sound of striking or hitting an object on another material. Each signal component corresponds to an excited eigen frequency, i.e., resonant mode, of the interaction between the object and the material. Each resonant mode has its own logarithmic damping characteristic, which is described by the amplitude envelope. The resonant modes convey properties of the object and material and properties of the force or type of the impact. Some properties of objects are mass, length, hardness, the amount of deformation (or bending) at the impact, and the measure of return to equilibrium after the impact. Gaver proposes that several of these properties can be described by sound synthesis parameters. Some studies have already shown that some type of information encoded by these parameters can be quite accurately conveyed (Buxton, Gaver, and Bly, 1992; Hermes, 1998). In gen-

eral, the following parameters can be controlled during the synthesis process; their correlations with real-world attributes are only suggested, not yet fully empirically tested.

- Varying the pattern of partial frequencies, i.e., composition of the sine waves, corresponds to the altering of the stroke object's shape;
- Varying the overall frequencies, i.e., the bandwidth, by adding or removing sinus components, corresponds to varying the size of the hit object;
- Varying the pattern of initial amplitudes, i.e., the steepness of the amplitude envelope at the onset of the sound, corresponds to the hardness of the hitting object;
- Varying the overall amplitude corresponds to the force or proximity of the hit;
- Varying the pattern of damping corresponds to the quality of the material, thus how the material deforms or returns to its original state.

For instance, wood, rubber, and metal impact sounds are differentiated by their decay rates; they produce qualitatively different deformation and recovery properties. In addition, the components of metal sounds are more pronounced by the existence of higher partials. Some preliminary empirical testing with the implemented synthesis (Hermes, 1998) has already identified some synthesis parameters to be coupled with the perception of materials. Glass impact sounds are characterized by partials with high centre frequencies (higher than 4 kHz) and exponential decay characteristics with a relatively small time constant (approximately 12.5 ms). Metal is characterized by partials with moderately high centre frequencies (around 1.6 kHz) and fast decay characteristics; the time constant is longer than 25-50 ms. Impact sounds of wood have partials with low frequencies (approximately 4 kHz). Their partials have long decays with a time constant of about 6.15 or 12.5 ms. Rubber sounds have even lower partials.

Not only understanding by what synthesis parameters information about the event is conveyed in sound, but also solid knowledge on physics, acoustics, psycho-acoustics, and signal processing pledge the way of generating natural sounding sounds. Subsequently, these natural sounds can be utilized as auditory icons in an interaction style which stands for events occurring in the interaction. What sound can do for an interaction style is documented elsewhere (Eggen, 1993; Gaver, 1997).

Besides knowledge on the synthesis parameters, some rules of thumb are helpful during sound design. It is often proposed that harmonic sounds are relatively obtrusive and annoy at long-term hearing or when the interaction style is used on a more regular basis. As a guiding principle, a sound in an interaction style is considered well-designed and appropriately utilized, when the user is not constantly aware of the sound, but notices, on the other hand, any sudden change or disappearance of the sound. Sinus components, for instance, sound 'less tonal' when shimmer is added to their centre frequencies, i.e., adding noise to their phase. Another strategy, which also renders other side-effects, is by adding sinus components with slightly inharmonic centre frequencies. To trade the burden of annoyance, one can randomly select synthesis parameters, e.g., centre frequencies, from a pool of 'allowable' parameters (or from some probability distribution) and synthesize the sounds accordingly. It results into slightly different versions of a 'prototypical' sound. Another remedy is to add small noise components to the sound.

7.1 Implementation

The synthesis algorithms are pioneered by Gaver (Buxton, Gaver, and Bly, 1992), but extended and implemented at the IPO as part of the Philips Ease-of-Use Research Programme for Sound and Vision 1997. Further extension and empirical testing of the algorithms is going on at the IPO (Hermes, 1998). The implementation, as documented here, is relatively primitive which results into the synthesis of sounds made by simple events (e.g. a single strike or hit). For instance, time varying sound generation is not possible. Although sound is an excellent medium to represent time varying events, the implementation does not yet allow interactive adaptation of the parameters during generation.

The class diagram is represented in Figure 19. The software hinges on the abstract class **Sound** which contains the implementation of signal-analytical operations in the time domain, such as addition, convolution, time-shifting, and multiplication. In addition, it declares the function *calcSound* which generates a signal component. All components such as periodic or noisy waveforms, but also envelopes, are derived from this abstract class and implement the function *calcSound*. The synthesis parameters, e.g., centre frequency, duration, are passed on at the creation of a signal component. Subsequently, generated signal components are first multiplied with an envelope component, before they are summed to arrive at a compound sound form. Finally, this sound form is convoluted with a pulse train.

The subclass **Bouncing** represents a pulse train which characterizes the repetitive impacts of a bouncing object. The amplitude of pulses decay and the time interval between two pulses decreases according to the 'loss of energy' law. In order to play back sampled audio, e.g. pre-fabbed speech waveforms, files according to the Windows' native sound format (WAVE or WAV format) (Kientzle, 1998) can be read in and handled as any other **Sound** object. The **Emphasis** subclass is a means to pre- or de-emphasis other sound components, i.e. altering the spectral slope.

In principle, the generated sound can be played back immediately. The sound generation however introduces some latencies due to computation. These latencies are only acceptable when the sound to be generated is not too complex, i.e., when it has a limited number of components and has a short duration. If the sounds need to be played *immediately* as prescribed by the client application, all sounds must be pre-generated during initialization of the client application, and stored at a place from which they can be retrieved easily. Sounds are therefore stored in an instance of a **Dictionary** class, referred to by the instance *SoundBook*; it acts like an ordinary hash table. When requested, sounds can be retrieved from this sound book, and played back with no latency.

Playing back non-speech sound (and sound originated from WAV-files) is done in a separate thread. The audio samples have to pass some wrappers before it reaches the DirectSound Dynamic Link Library (see also Chapter 6 for streaming audio). If several audio output formats are used simultaneously at the DirectSound level, it is recommended that all audio data comply to a uniform format to achieve an optimal output quality. Therefore, sample rate conversion is required before the impact sounds are played back by the audio device. On request of a client, impact sounds can be converted and subsequently stored in the soundbook. Or if not so, the **AudioDeviceWrapper**

object converts the audio, if necessary. However, the latter possibility introduces some delay.

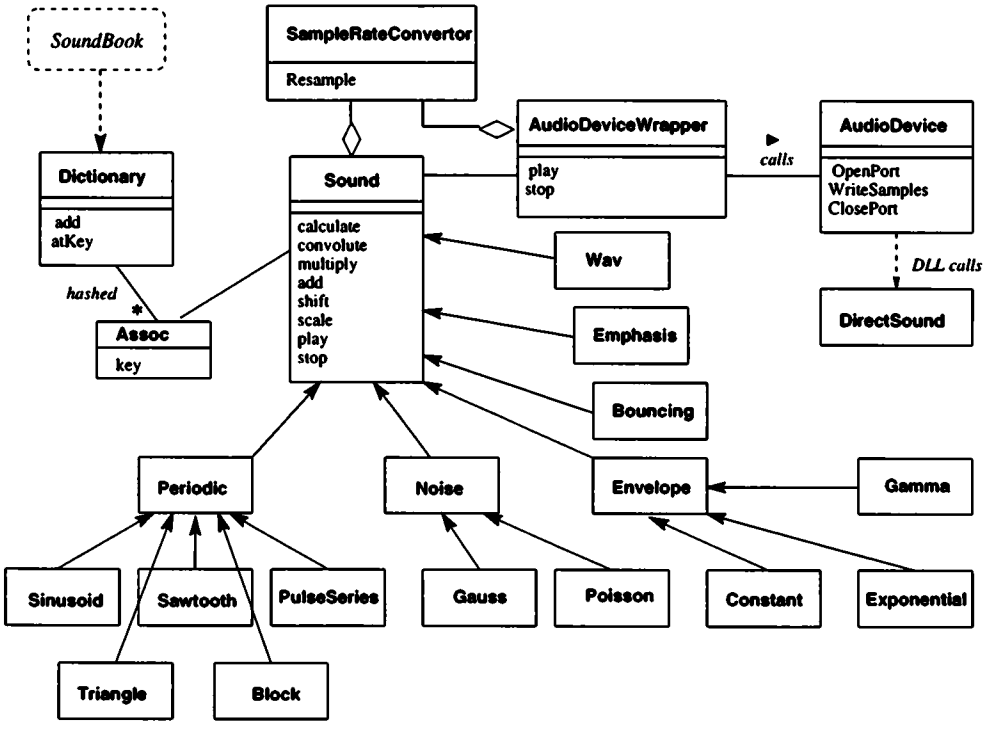

Figure 19. The class diagram of the Constrained Additive Synthesis technique for the generation of impact sounds.

7.1.1 Specification file

The whole generation process is determined by a specification file. This file is read and parsed, the required components are created with the specified parameters, and the generation process starts off.

The following abbreviations are used for identifying the periodic signal components:

`sin` - sinus component,
`tri` - triangle component,
`saw` - sawtooth component,
`blk` - block component,
`pls` - pulse series component,
Their parameters are the following: `sf` - sampling frequency, `cf` - centre frequency, `ph` - phase, `am` - amplitude, `du` - duration, `mf` - modulation frequency, `mi` - modulation index, `sis` - sigma (std.dev.) for shimmer distribution, `sij` - sigma (std.dev.) for jitter distribution, and `se` - seed of random generator.

The following abbreviations are used for identifying noise components:

poi - Poisson noise component with parameters sf - sampling frequency, se - seed of random generator, cf - lambda of Poisson distribution, am - amplitude, and du - duration.

gau - Gaussian noise component with parameters sf - sampling frequency, se - seed of random generator, si - sigma (std.dev.) of Gaussian distribution, and du - duration.

The following abbreviations are used for the envelopes

gam - gamma envelope with parameters sf - sampling frequency, ga - gamma constant, tc - time constant, am - amplitude, and du - duration.

exp - exponential envelope with parameters sf - sample frequency, tc - time constant, am - amplitude, and du - duration.

con - constant envelope with parameters sf - sample frequency, am - amplitude, and du - duration.

Abbreviation for the two remaining components are

emp - (pre- or de-) emphasis component with parameters sf - sampling frequency, cf - lambda of emphasis, se - seed for random generator, am - amplitude, md - modulation depth, mf - modulation frequency, and du - duration.

bou - bouncing pulse sequence with parameters sf - sampling frequency, si - start interval of the bounce (negative value indicates a reversed bouncing sequence), am - amplitude, and du - duration.

The following example is a specification of a sound which is generated by adding three sinus component with rather low centre frequencies multiplied with a rather rapidly decaying gamma envelope with a slow onset, and convoluted by a single pulse.

```
sin -cf 150 -ph 0.0 -du 0.05
gam -ga 2 -tc 0.005 -du 0.05
sin -cf 160 -ph 0.5 -du 0.05
gam -ga 2 -tc 0.005 -du 0.05
sin -cf 170 -ph 0.0 -du 0.05
gam -ga 2 -tc 0.005 -du 0.05
pls -cf 1 -am 30 -du 0.001
```

The result of the generation is shown in Figure 20. It sounds like a rubber object hit by some other solid object.

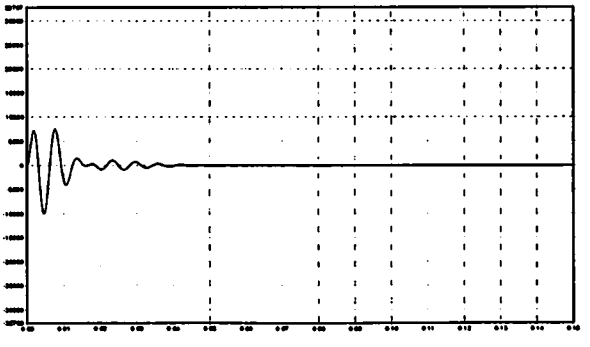

Figure 20. The generated sound of a rubber object hit by some other solid object.

Another example is an impact sound made up of a larger set of sinus components with high centre frequency and rather slowly decaying gamma envelopes with an immediate onset.

```
sin -cf 4000 -ph 0.0 -du 0.15
gam -ga 1 -tc 0.02 -du 0.15
sin -cf 4200 -ph 0.0 -du 0.15
gam -ga 1 -tc 0.02 -du 0.15
sin -cf 4300 -ph 0.0 -du 0.15
gam -ga 1 -tc 0.02 -du 0.15
sin -cf 4300 -ph 0.0 -du 0.15
gam -ga 1 -tc 0.02 -du 0.15
sin -cf 5200 -ph 0.5 -am -1 -du 0.15
gam -ga 1 -tc 0.02 -du 0.15
pls -cf 1 -am 30000 -du 0.001
```

The result of the generation is shown in Figure 21. It sounds like a glass object hit by some other solid object.

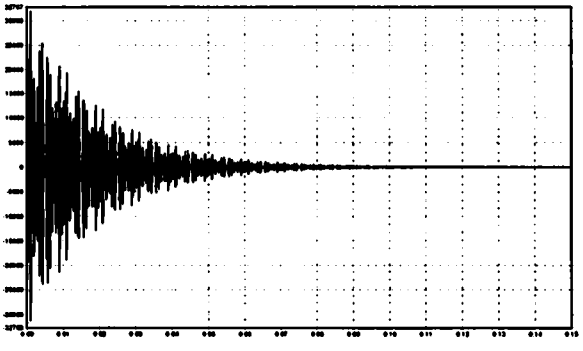

Figure 21. The generated sound of a glass object hit by some other solid object.

7.1.2 Interface at client application level

The Constrained Additive Synthesis algorithms are compiled and linked into a Dynamic Link Library (DLL) which can be used in applications, preferably running under Visual Basic. Impact sounds or WAV sounds are referred to by an index, i.e., a hash value, which allows to read specification or WAV files and store the sounds internally, to convert their sampling rate or stereo/mono format, and to play back or stop playing back the sounds. Two additional interface functions indicate whether an impact or WAV sound can be player or not. The following DLL interface is defined.

```
int WINAPI ImpactSnds_AddImpactFile (char *index, char *fileName);
int WINAPI ImpactSnds_AddWavFile (char *index, char *fileName);
int WINAPI ImpactSnds_PlayImpact (char *index);
int WINAPI ImpactSnds_StopImpact (char *index);
int WINAPI ImpactSnds_ConvertSampRate (char *index, int sampRate,
           int numChannels);
void WINAPI ImpactSnds_SetWavPlayable (int on);
void WINAPI ImpactSnds_SetImpactPlayable (int on);
```

By declaring this interface at Visual Basic, these interface functions can be directly accessed in a Visual Basic application.

7.1.3 Source code example

The following Visual Basic code excerpt plays back a pre-generated impact sound or a wav-file sound at the same audio output format.

```
' Some declarations from the ImpactSound DLL
Declare Function ImpactSnds_AddImpactFile Lib "ImpactSound" _
        (ByVal key As String, ByVal fname As String) As Long
Declare Function ImpactSnds_AddWavFile Lib "ImpactSound" _
        (ByVal key As String, ByVal fname As String) As Long
Declare Function ImpactSnds_PlayImpact Lib "ImpactSound" _
        (ByVal key As String) As Long
Declare Function ImpactSnds_ConvertSampRate Lib "ImpactSound" _
        (ByVal key As String, ByVal sampRate As Long, _
         ByVal numChannels As Long) As Long

Private Sub Form_Load()
      ImpactSnds_AddImpactFile "wood", "Wood.imp"
      ImpactSnds_ConvertSampRate "wood", 16000, 1
      ImpactSnds_AddWavFile "bill", "C:\\windows\\media\\The Microsoft Sound.wav"
      ImpactSnds_ConvertSampRate "bill", 16000, 1
End Sub

Private Sub Command1_Click()
      ImpactSnds_PlayImpact "wood"
End Sub

Private Sub Command2_Click()
      ImpactSnds_PlayImpact "bill"
End Sub
```

8 Software for playing MPEG Audio

Music audio is compressed according to the MPEG-1 Layer II audio format (stereo, 125 kbps). MPEG stands for Moving Pictures Expert Group, a consortium formed by the ISO committee to develop a standard way to compress high-quality video and audio sequences. As needs from a variety of industry converges, this has resulted in various MPEG standards for applications ranging from satellites to consumer electronics. New standardization processes are still going on. An introductory to MPEG Audio is presented by Pan (1995) and an implementation of MPEG-1 (de)compression is published by Kientzle (1998). The World Wide Web is also a valuable resource on MPEG standards. (e.g., http://drogo.cselt.stet.it/mpeg/ or http://www.mp3.com/).

The MPEG standard comprises three layers; each layer provides a successively better quality at the cost of a more complex and computationally intensive implementation. For this project, Layer II was considered a compromise between storage needs, computational needs, and accepted quality loss. Dependent on the audio content, compression factors for MPEG-1 Layer II higher than 10 are common for music which is originally sampled at CD quality (44.1 kHz, 16 bit precision, stereo).

8.1 Implementation

A complete explanation how the MPEG Audio Player works is beyond the scope of this document. Moreover, the core of the decoder is based on loosely documented freeware software, known as maplay. This freeware decoder is however a computationally efficient C++ implementation and manages to decode all layers of the first two MPEG standards. It can be compiled on a multitude of platforms (e.g. SPARC, AIX, HPUX, Linux, Windows 32). The software is allowed to be (re)distributed under the GNU General Public License as published by the Free Software Foundation.

A global overview of MPEG audio decoding works as follows. A MPEG bitstream consists of frames of compressed data. Each frame consists of a 32-bits frame header that defines the format of the data in that frame. Decoding MPEG data is now tracking the bitstream, identifying and parsing the frame headers, and use the information in the headers to decompress individual frames into pulse code modulation (PCM) audio data.

The MPEG Audio Player utilizes the freeware decoder as a black box in which the bitstream is inputted and audio as PCM samples is outputted. A **MPEG_Args** class serves as a high-level control mechanism for controlling both the decoding process and the playing of PCM data. The MPEG Player is, for obvious reasons, multi-threaded. The client occupies the main thread, which starts up a designated decoding thread which issues PCM samples, which are, in turn, played back in a separate thread.

As shown in Figure 22, the **MPEG_Args** class is an 'ennobled' C structure, that facilitates control from the outside. Therefore, it has flags such as pause/resume, stop, fastforward, and fastbackward that can be set by the client. Setting a flag can be done while underlying decoding and play threads are running. Therefore, flag-setting is protected

by using mutual exclusion, i.e. mutex, techniques. After setting the flags, underlying threads react accordingly.

The **MPEG_Args** class has also a reference to the bitstream and maintains the current frame header. An instance of this class is fed into a decoding thread which utilizes the bitstream. At the other side of the decoding process, an abstract class **OBuffer** declares two functions called *append* and *write_buffer*. Derived subclasses of **OBuffer** facilitates playing back audio samples on different platforms and audio card peripherals. Our MPEG Player uses an instance of the **Adev_OBuffer** class that controls DirectSound. The decoding thread has an instance to this class and *appends* PCM samples when decoded. After a frame is decoded, it invokes a *write_buffer* operation notifying a frame is completely decoded. However, **Adev_OBuffer** buffers more data internally; DirectSound recommends buffers comprising, at least, 1 second of data to be passed on without hearing noticeable artefacts (see also Chapter 6 for streaming audio). For a fine control, **Adev_OBuffer** issues small chunks of data in a separate thread to the **AudioDevice**.

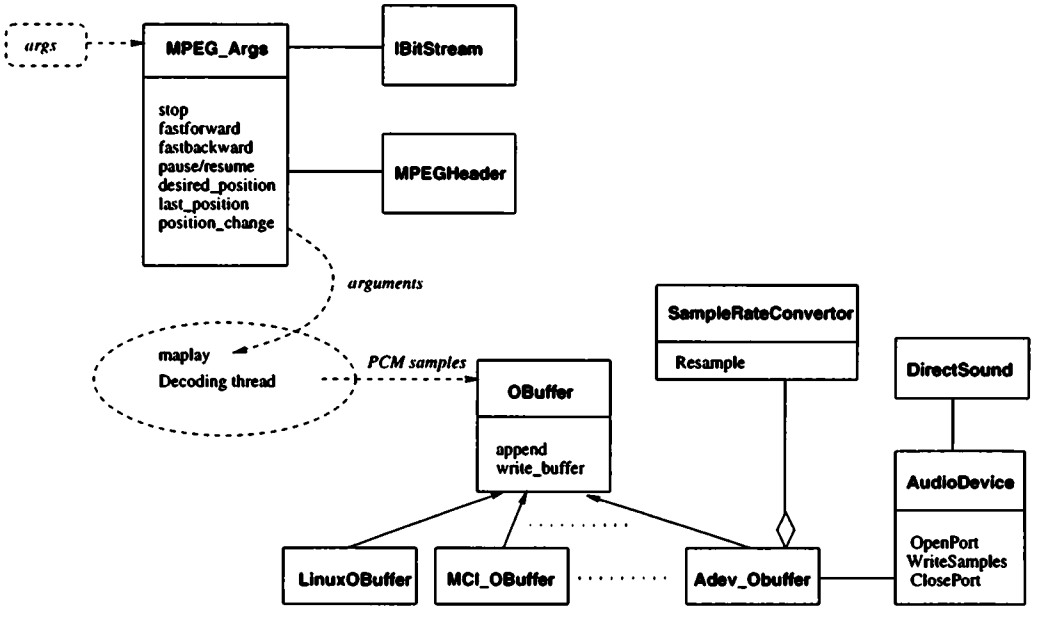

Figure 22. The class diagram of the essentials of the MPEG Audio Player.

8.1.1 Interface at client application level

The MPEG Audio Player is compiled and linked into a Dynamic Link Library (DLL) which can be used in applications, e.g., running under Visual Basic. A client that wants to use the DLL interface of the MPEG Player needs to work as follows. First of all, the MPEG player has to be informed about an incoming MPEG bitstream, which is somewhere located as a file, by invoking `MPEGPlayer_SetMPEGPlayer`. Subsequently, this bitstream can be decoded and made audible directly by the designated play function. Decoding and playing can be immediately stopped, or temporarily paused to be resumed later on. In addition, the playing functionality can be forwarded or rewinded

at a fast pace. The current playing state of the MPEG Audio Player can be peeked at all times, if necessary.

As the decoding process proceeds frame-by-frame, a client can ask for the current frame that is currently processed by calling `MPEGPlayer_CurrentFrame`. To get an indication how far the decoding process has already reached, the client can assess the current frame against the last frame in the bitstream. The function `MPEGPlayer_LastFrame` returns the number of frames in the bitstream which indicates the total length of the corresponding file.

Frames can be spontaneously skipped back and forth during the decoding process by using `MPEGPlayer_SetPlayPos`, while playing back MPEG audio. Its argument `playpos` refers thus to a frame number. Facilities are built in to memorize the position of a frame per bitstream. This frame position (e.g. indicating a starting play position) can be recalled, when needed, or it can be simply 'forgotten'. It must be emphasized that only five frame numbers (i.e., playpos) are memorized in a cyclic manner, implying that each new sixth frame number overrides the first frame number. Also, only one frame number per bitstream is memorized; each new frame number of a bitstream overrides the previous one. Lastly, bitstream (i.e., MPEG file names) are referred to by an integer value called `trno`. The coupling between file names and such integer values need to be maintained by the client application.

```
int WINAPI MPEGPlayer_SetMPEGFile ( char *fileName );
int WINAPI MPEGPlayer_Play ();
int WINAPI MPEGPlayer_Stop ();
int WINAPI MPEGPlayer_Pause ();
int WINAPI MPEGPlayer_Resume ();
int WINAPI MPEGPlayer_isPlaying ();
int WINAPI MPEGPlayer_isPaused ();

int WINAPI MPEGPlayer_StartFastForward ();
int WINAPI MPEGPlayer_StopFastForward ();
int WINAPI MPEGPlayer_StartFastBackward ();
int WINAPI MPEGPlayer_StopFastBackward ();
int WINAPI MPEGPlayer_isFastForwarding ();
int WINAPI MPEGPlayer_isFastBackwarding();

int WINAPI MPEGPlayer_CurrentFrame ();
int WINAPI MPEGPlayer_LastFrame ();

int WINAPI MPEGPlayer_SetPlayPos ( int playpos );
void WINAPI MPEGPlayer_MemorizePlayPos ( int trno, int playpos );
int WINAPI MPEGPlayer_RecallPlayPos ( int trno );
void WINAPI MPEGPlayer_ForgetPlayPos ( int trno );
```

8.1.2 Source code example

The following Visual Basic code excerpt utilizes the MPEG Audio player. It simply toggles between playing and stop playing a piece of 'Miles Davis' MPEG audio. Playing can also be fast-forwarded.

```
' Some declarations from the MPEGPlayer DLL
Declare Function MPEGPlayer_SetMPEGFile Lib "MPEGPlayer" _
        (ByVal fname As String) As Long
Declare Function MPEGPlayer_Play Lib "MPEGPlayer" () As Long
Declare Function MPEGPlayer_Stop Lib "MPEGPlayer" () As Long
Declare Function MPEGPlayer_isPlaying Lib "MPEGPlayer" () As Long
```

```
Declare Function MPEGPlayer_StartFastForward Lib "MPEGPlayer" () As Long
Declare Function MPEGPlayer_StopFastForward Lib "MPEGPlayer () As Long
Declare Function MPEGPlayer_isFastForwarding Lib "MPEGPlayer" () As Long

Private Sub Command1_Click()
      If (Not MPEGPlayer_isPlaying) Then
         MPEGPlayer_SetMPEGFile "Miles Davis.mp3"
         MPEGPlayer_Play
      Else
         MPEGPlayer_Stop
      End If
End Sub

Private Sub Command2_Click()
      If (Not MPEGPlayer_isForwarding) Then
         MPEGPlayer_StartForward
      Else
         MPEGPlayer_StopForward
      End If
End Sub
```

9 Software for visualizing a roller

One of the 'perceived affordances' of a ball is its possibility to roll or rotate it along one of its axes. To make use of this provision and to reinforce the interaction of scrolling through a list of items, the list is virtually tapered round the ball, as shown in Figure 23. Intuitively, this list is then visually represented as a roller. If the appropriate facilities are built in, it is possible that any forward or backward ball rotation induces an immediate corresponding scroll in the list. Force feedback and the sound of clicks might even bring along a further immersion to the simple task of scrolling. Other actions on the ball, such as pushing down the ball, can be programmed to cause the selection of an item. If, in addition, the list is extended into a two-dimensional matrix of items, ball rotations to the left or right might correspond to wandering along the columns of the matrix.

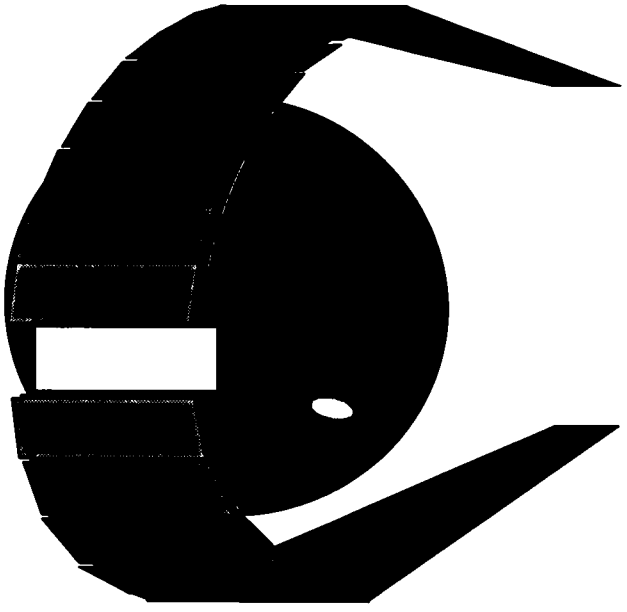

Figure 23. An infinitely long list of items is virtually tapered along a ball.

The SelectionBall ActiveX control visualizes an indefinite list (or matrix) of items as a cylindric roller. Properly speaking, it is capable of maintaining more than one list (matrix) at the same time. The client application indicates what list is displayed and what list is thus under its direct control. SelectionBall only renders that part of the displayed list that is currently at the front, and highlights the item that has the input focus. At any moment, items can be added at any place in the list or removed from any place in the list. Consequently, the visual representation of the roller will adapt itself to this new situation. The SelectionBall does not provide any other output modality than the visual one. The integration of sound, speech, or force feedback have to be dealt with at the client application level. Although the SelectionBall was designed with having the trackball in mind, there is nothing against using it with another input modality or device.

9.1 Implementation

As might be expected after reading Chapter 6, the SelectionBall is not based on DirectX technology. Its visualization is based on the Windows 32 GDI graphics API.

As shown in Figure 24, the implementation consists of two parts. A dynamic link library (dll) does the real job of visualization, list maintenance, and carrying out all requests originating from the client application. The ActiveX part only interfaces between the client application and the dll part; all client calls are directly delegated to the dll. In fact, each instance of a **SelectionBallControl** has its own instance of a **SelectionBall**.

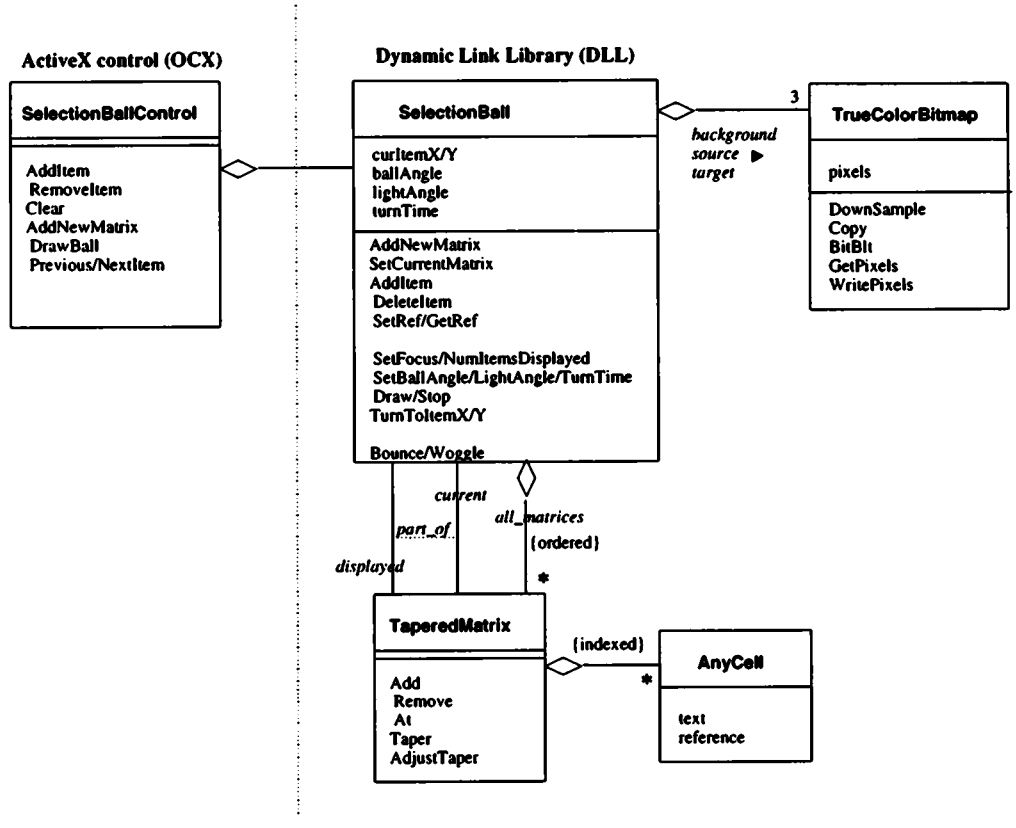

Figure 24. The SelectionBall consists of a ActiveX and a DLL part. All 'intelligence' is kept in the DLL.

An instance of **SelectionBall** class takes care of mapping, i.e., wrapping, matrices on curved bitmap representations. The necessary Windows 32 GDI API calls are encapsulated in the class **TrueColorBitmap**. This bitmap class declares and implements the operations for manipulating bitmaps, such as creating bitmaps, copying whole bitmaps, or bit block transferring (bit-blt) from one area to another within a bitmap. The bitmap class also declares a *DownSample* method, which applies a median filter on its instance; median filtering is required to avoid undesired blurring effects, i.e., aliasing effects, due to wrapping a rectangular bitmap around a curved surface. By calling *WritePixels*, the content of the bitmap is visualized in a so-called device context, a programmer's handle

to freely access the GDI functions and data structures in a window. All bitmap calculations are done with Microsoft's TrueColor RGB colour model.

As shown in Figure 24, the **SelectionBall** class has references to three instances of the **TrueColorBitmap** class. The *background* instance refers to a background colour representation which suggests the sphericity of the visual representation by using an illumination technique. The illumination is dependent on the angle of an incident light beam, i.e., *lightAngle*. The *source* instance contains some textual information that must be wrapped. The *target* instance refers to the final result consisting of the overlay of the wrapped source bitmap on the background bitmap.

The **SelectionBall** class can maintain any number of matrices instantiated from the class **TaperedMatrix**. The **TaperedMatrix** class is essentially an ordinary matrix with holds references to instances of the **AnyCell** class. The matrix class simply allocates memory when new instances (i.e., new columns and rows) are added and frees memory, if it is allowed to do so. In addition, it declares two function *Taper* and *AdjustTaper* which will be described later on. The class **AnyCell** contains a textual description and an integer value called *reference*, which can be used by the client application. In the sequel, the terms 'list' and 'matrix' are used interchangeably to denote an instance of the class **TaperedMatrix**.

The lists are built up by adding all items one-by-one by the client application while switching from one list to the other. Adding a new list is done by calling *AddNewMatrix*, which can be subsequently filled with items. Switching to another list is done by *SetCurrentMatrix*. The **SelectionBall** class holds a reference to the *current* matrix. As the lists are kept in the same order as they are created, the client application should memorize the sequence number of each list (which is also returned by *AddNewMatrix*). Adding an item is done by the member function *AddItem*. As the items are positioned in a two-dimensional matrix, a x and a y coordinate are required in the function call. Along with the coordinates, a text string is passed in the argument list. This text string is the item's first line of text that will be displayed. Subsequent calls to *AddItem* with the same coordinates, thus referring to the same item, do not override previously passed lines of texts. They add, on the other hand, extra lines of text to the item. As items refer to some semantics at client application level (e.g., a piece of music), it is possible to attach a integer value that refers to the client's semantics. By calling *SetRef*, the client can attach a specific integer value to an item. By calling *GetRef*, the client can recollect the integer value. Items can be simply deleted from the *current* list by *DeleteItem*. For clarity, items are always referred to by their x and y coordinates. Preserving any list order (e.g., alphabetical) is the responsibility of the client application.

The core task of **SelectionBall** class is wrapping the items' text on a curved surface, render this representation, and updating the representation, when needed. The best way to convey the process of wrapping the list is shown in Figure 25. The wrapping process consists of three stages: determining a running window, writing the content of the running window onto a bitmap, and wrapping this bitmap.

The **SelectionBall** class maintains a so-called running window over the indefinitely long *current* list. This window, which is referred to by the *displayed* pointer (see Figure 24), keeps track what part of the entire list is candidate to be wrapped. At initialization, the running window is determined by the *Taper* member function of the **TaperedMatrix**

class. Any adjustments to the running window (when the roller scrolls) are determined by the *AdjustTaper* function. As a reference point, both tapering functions need to know what current item is currently in focus, i.e. is at the very front of the roller. For that item is held at the middle of the running window. The item in focus is maintained by the **SelectionBall** class (*curItemX* and *curItemY*). The size of the running window can be set by *SetNumItemsDisplayed*, and defines thus how many items are wrapped around the roller.

The textual information of the items in the running window are written onto a 'flat' bitmap referred to by *source*. Subsequently, this bitmap is subjected to a warp function and superimposed on the *background* bitmap to arrive at a roller representation in the *target* bitmap.

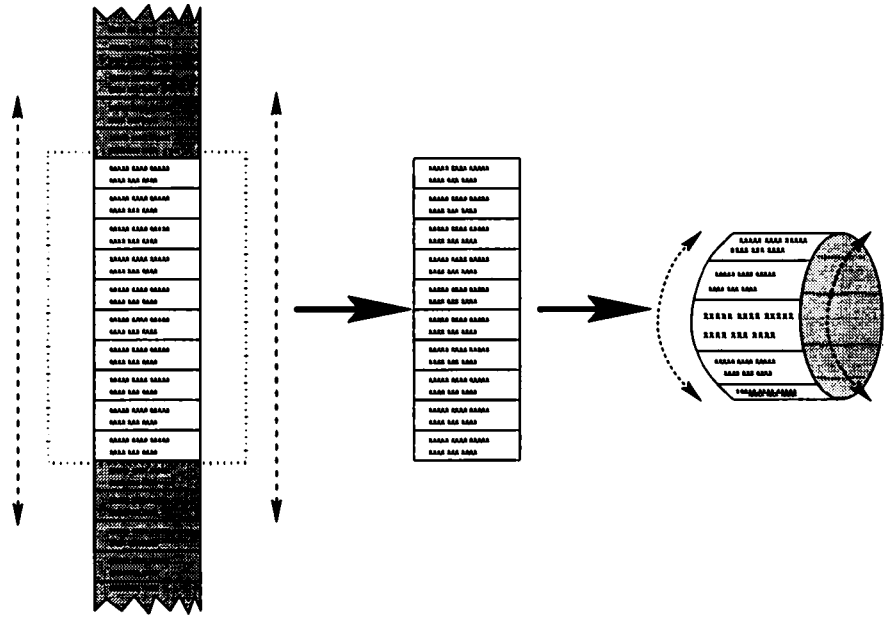

Figure 25. A running window is placed on an indefinitely long list of items. This window determines what portion of the list must be tapered along a curved surface. For visualization, that part of the list is 'cut out' of the entire list before it is actually wrapped by 'gluing together' both ends of the small list. Scrolling up and down the entire list now corresponds with rotating up and down a roller.

Some details of the warp function are depicted in Figure 26. Analogue to enfolding a sheet of paper around some cylindric object, the two ends of a rectangular bitmap come together at the back of the object; the sight of the viewer is taken as the view plane as shown in Figure 26. This implies that only half of the wrapped bitmap, i.e. half of the items in the list, is visible from the viewer's point. Thus, half of the original *source* bitmap has to be projected upon the *target* bitmap. As shown in Figure 26, the variable *ballAngle* indicates the displacement at what point, i.e., scanline, in the original *source* bitmap the warping starts. Warping now means adding the *mean* of a bundle of scanlines from the *source* bitmap onto a single scanline of the *target* bitmap. The size of the bundle is determined by an running *arccosinus* function. However, taking the mean of

several bitmap pixels causes considerable blurring, i.e., aliasing, in the warped representation. Therefore, the warping transformation is done with bitmaps that are a factor (e.g., 2, 4) bigger than the original bitmaps. After the transformation, the warped bitmap is down-sampled to its original proportions. The item with the current input focus is now at the very front of the roller and will be highlighted.

Several visual effects are implemented within the **SelectionBall** class: a continuous roll movement, bouncing and woggling. It is not hard to imagine that these effects are implemented by adding repetitively small offsets to the variable *ballAngle*, before the bitmap is warped and rendered. A continuous roll movement is used when member function *TurnToItemY* is called; it puts a given item in focus at the front of the roller. The function *TurnToItemX* is provided when wandering along the columns of a matrix. The duration of the roll movement when turning from one item to another can be specified by *SetTurnTime*.

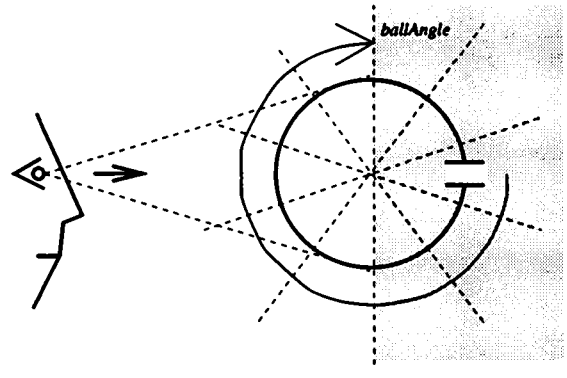

Figure 26. The variable *ballAngle* refers to the displacement at what point the wrapping starts. At the back of the virtual roller, both ends of the 'running' window are glued together.

As it takes some time to bring the visual effects to full prosperity, all member functions try to start up a new thread and are asynchronous in nature. To some extent, the client application can break up the current thread within a **SelectionBall** instance by calling *Stop*. Only a single thread is allowed to be active within an instance of **SelectionBall**; both code and address space are shared among multiple threads. In other words, a member function that wants to create a thread within an object have to wait on other threads, i.e., member functions, within that same object to complete. As this waiting is done in the main thread, i.e., the processing time of the client application, the client application's ability to break into threads is limited.

9.1.1 Interface at client application level

The SelectionBall is compiled and linked into a Dynamic Link Library (DLL). Although this library can be used in client applications by itself, an ActiveX control interface is wrapped around it that facilitates the integration into a Visual Basic application. The number of public properties and subroutines is quite numerous. The following public properties are defined.

```
' Properties for rendering the roller
' Background color
Public Property Get BackColor() As OLE_COLOR
Public Property Let BackColor(ByVal New_BackColor As OLE_COLOR)
' Incident light beam angle for illumination technigue
Public Property Get LightAngle() As Double
Public Property Let LightAngle(ByVal New_LightAngle As Double)
' The font type of the texts on the roller
Public Property Get FontName() As String
Public Property Let FontName(ByVal New_FontName As String)
' The intensity in which the item in focus is displayed
Public Property Get FocusLight() As Long
Public Property Let FocusLight(ByVal New_FocusLight As Long)
' The displacement angle from which wrapping starts off
Public Property Get BallAngle() As Double
Public Property Let BallAngle(ByVal New_BallAngle As Double)
' Parameters for deblurring the wrapping
Public Property Get AntiAliasing() As Boolean
Public Property Let AntiAliasing(ByVal New_AntiAliasing As Boolean)
Public Property Let UpSample(up As Integer)
' The size of the running window that is wrapped
Public Property Get NumItemsDisplayed() As Long
Public Property Let NumItemsDisplayed(new_NumItemsDisplayed As Long)
' The size of the bitmap (i.e. control) in pixels
Public Property Get width() As Long
Public Property Get height() As Long

' Returns what item is currently in focus
Public Property Get curItemX() As Long
Public Property Get curItemY() As Long

' Time it takes for a roll movement
Public Property Let TurnTime(ByVal newTime As Double)
Public Property Get TurnTime() As Double

' Management for the tapered matrices (i.e. lists)
Public Property Get CurrentMatrix() As Long
Public Property Let CurrentMatrix(ByVal New_CurMatrix As Long)
Public Property Get NumMatrices() As Long
' Number of rows and columns in the current matrix (i.e. list)
Public Property Get NumItemsY() As Long
Public Property Get NumItemsX() As Long
' Remember and recall the item in focus when matrices are switched
Public Property Let Memorize(ByVal New_Memorize As Boolean)
Public Property Get Memorize() As Boolean

' Returns whether the control is busy with drawing
Public Property Get IsBusy() As Boolean
```

A client application can built up the content of the matrices by invoking the appropriate public subs.

```
Public Sub AddNewMatrix()
Public Sub AddItem(ByVal x As Long, ByVal y As Long, ByVal str1 As String,_
        Optional redraw As Boolean)
Public Sub RemoveItem(ByVal x As Long, ByVal y As Long, _
        Optional redraw As Boolean)
Public Sub SetRef(ByVal x As Long, ByVal y As Long, ByVal aux As Long)
Public Function GetRef(ByVal x As Long, ByVal y As Long) As Long
Public Sub Clear()
```

Manipulating the roller by turning over to the next item on the roller can be at best invoked by the following public subs. The x coordinate defines the columns of the

matrix, whereas the y coordinate defines the rows of the matrix. If only a one-dimensional matrix, i.e., a list, is used, only `PreviousItemY()` and `NextItemY()` need to be called.

```
Public Sub NextItemX()
Public Sub PreviousItemX()
Public Sub NextItemY()
Public Sub PreviousItemY()
```

Additionally, a client application can invoke calls to woggle or bounce the roller. The `woggle` routine requires two scale values (between -1.0 or 1.0) that specify by what negative or positive offset the wrapping should start. The boolean `bWoggleAll` specifies whether the whole roller or only a portion should be moved. The `bounce` routine requires a `startInterval` that specifies the time interval between the first two bounces, and the total `duration` of the bounce. The `nowait` boolean specifies whether bouncing should occur asynchronously. `DrawBall` simply displays the roller. `StopBall` stops the current thread within the SelectionBall control.

```
Public Sub Woggle(ByVal inx As Double, ByVal iny As Double, _
        ByVal bWoggleAll As Boolean)
Public Sub Bounce(ByVal startInterval As Double, ByVal duration As Double, _
        ByVal nowait As Boolean)
Public Sub DrawBall()
Public Sub StopBall()
```

9.1.2 Source code example

The following excerpt of Visual Basic source code utilized the ActiveX control of the SelectionBall. Besides some command buttons and a textbox, it is assumed that an instance of the control is dropped onto the VB form and can be referred to as `SelectionBallControl1`. The example first fills the control with three lists of 20 items each. By clicking on the command buttons, one can cycle through the lists, or go to other items within a list. One command button displayes status information about the control in a textbox.

```
Private Sub Form_Load()
      Dim l as Long
      Dim y as Long
      Dim str as String

      For l = 1 To 3
         For y = 0 To 19
               str = "Item " & y+1
               SelectionBallControl1.AddItem 0, y, str, False
               SelectionBallControl1.SetRef 0, y, y+1
         Next y
         If l < 3 Then
               SelectionBallControl1.AddNewMatrix
         End If
      Next l
End Sub

Private Sub Command1_Click()
      SelectionBallControl1.NextItemY
End Sub

Private Sub Command2_Click()
      SelectionBallControl1.PreviousItemY
```

```
End Sub

Private Sub Command3_Click()
      Dim newl as Long
      Dim maxl as Long

      newl = SelectionBallControl1.CurrentMatrix + 1
      maxl = SelectionBallControl1.NumMatrices
      If newl >= maxl Then
         newl = 0
      End If
      SelectionBallControl1.CurrentMatrix = newl
End Sub

Private Sub Command4_Click()
      Dim str as String
      Dim curl as Long
      Dim itx as Long
      Dim ity as Long
      Dim ref as Long

      curl = SelectionBallControl1.CurrentMatrix
      itx = SelectionBallControl1.CurItemX
      ity = SelectionBallControl1.CurItemY
      ref = SelectionBallControl1.GetRef(itx,ity)
      str = "Current list number " & curl & VbCrLf
      str = str & "Item in focus " & itx & " " & ity & VbCrLf
      str = str & "Reference " & ref
      TextBox1.Text = str
End Sub
```

10 Evaluation of a multimodal interaction style for music programming

In this chapter, an experimental evaluation is presented of the usability properties of a multimodal interaction style for music programming. The design and implementation of the interaction style are reported in preceding chapters. The evaluation concentrates, in particular, on the presence or absence of a visual display combined with a tactual and auditory interface.

10.1 A multimodal interaction style

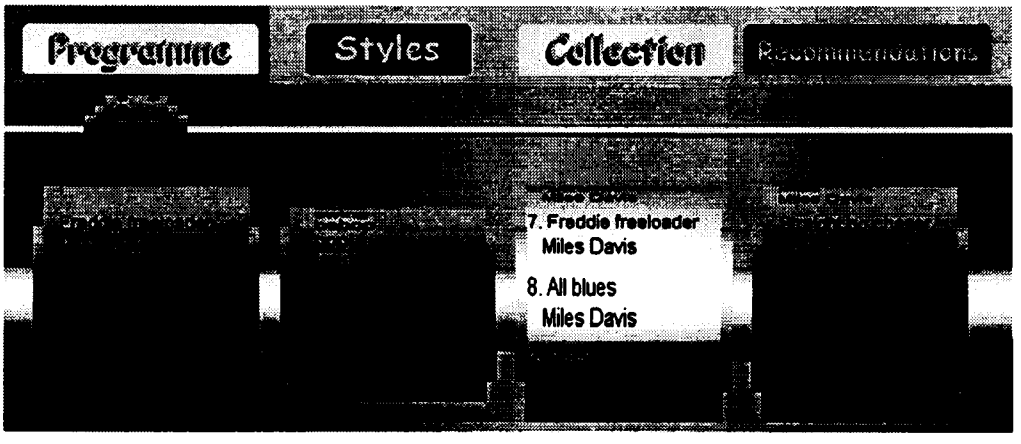

Figure 27. The visual display of the interface.

An interaction style is defined as the specific mode of communication that exists between a user and an interactive device. The central theme of an interaction style is its conceptual model for interaction. As already discussed in Chapter 2, the conceptual model was refined by an iterative design-and-evaluation process before it was subjected to the user experiment. Iteratively, two different users worked with the interaction style, and imperfections pertaining to usability were subsequently repaired. If neither user was able to acquire such proficiency after five minutes of free exploration that he/she could perform a given music programming task, it was concluded that learnability was insufficient. Users spontaneously made suggestions for improvement. No more than five iterations, i.e., 10 users, were necessary to master the interaction style; also, no more important shortcomings or user complaints were reported. It was concluded that no further refinements were considered necessary.

The final interaction style evolved into a multimodal interaction style for music programming, in which three output modalities (visual, auditory, and tactual) are combined. The conceptual model consists of a metaphor or physical analogy of a slot machine equipped with rollers. As shown in Figure 27, the visual display of the interaction style contains four main rollers, on which the music programme, the music styles, the music collection, and the music recommendations are projected. In the style, user control of the interaction proceeds entirely by manipulating the IPO force feedback

trackball (Engel, Haakma, and van Itegem, 1990). The force feedback trackball was used to minimize the number of control elements without sacrificing issues of effectivity and efficiency. Essentially, the trackball can be moved in two dimensions, i.e., in lateral and back/forward directions. In addition, the trackball can be pressed. By backward and forward trackball movements, rollers can be set in motion to bring another item at the front. A small hand movement brings the next item to the front of the roller. A somewhat faster hand-stroke covering a somewhat longer distance skips the next two items. By lateral trackball movements, one can hop from one roller to another roller. The tactual force feedback mediated by the trackball conveys the feeling of setting rollers into motion or jumping from one roller to the other; for instance, the discrete steps of a roller movement feel like bulges and notches while carrying out the movement. By using the roll movements, a user can choose a particular music style on the music styles roller and search further for pieces of music within that style on the music collection and recommendation rollers. A single press on the trackball, i.e. single click, evokes information about the current music style to be uttered. Double-pressing the trackball, i.e., double-clicking, means adding or removing a piece of music to or from the music programme.

Rotating the roller step-wise also produces audible clicks. Larger roller movements produce a rattling sound. Synthetic speech feedback informs the user about states of the interaction; the user is informed about which roller is currently in focus, how many pieces of music are added to the music programme, what music style is selected, etc.

10.2 Learning to operate an interactive device

When users use an interactive device for the first time to attain a certain goal, they first have to master, at least, the goal-relevant parts of the device's interaction style. An interaction style is essentially a technological product that confronts users; it defines the specific mode of communication between user and device. In order to comprehend the interaction style, user may develop an internal representation, i.e., mental model, of the interactive device. As different mental models can give rise to different user behaviour, it may be important that users assimilate the default conceptual model of the interaction style as their mental model. In the absence of visual display of information, it is assumed that first-time user have to resort to explorative behaviour using other nonvisual sensory modalities to construct a mental model. In that case, providing the default conceptual model to users may be helpful to attain goals efficiently. In general, it is often observed that novices are unfamiliar or do not recognize, at least, parts of the interaction style, as being relevant for attaining their goal. Especially in domains in which users should perceive at a glance the goal-relevant parts of the interaction style such as public electronic interactive services and consumer electronics, learnability is a fundamental usability issue. Learning to operate an interactive device is essentially the acquisition of a skill, i.e., the coordinated behaviour between perception, cognition, and motor action. It is commonly agreed that skill acquisition progresses through stages or involves several stages simultaneously before users become skilled performers or experts of an interactive device.

10.2.1 Mental model

It is generally believed that users assimilate some form of internal representation, i.e., mental model, of an interactive device which explains for them the device and helps them to reason about procedures for controlling the device (Carroll and Olson, 1988; Young, 1981; Halasz and Moran, 1983; Staggers and Norcio, 1993). Such a mental model encompasses a individual conception of the interactive device, and how one should deal with it. It is thought of containing essentially personal declarative and procedural knowledge of a particular domain (Gentner and Stevens, 1983).

Because there is confusion about the terms in this area of research, we adopt the definitions for which the names were re-arranged by Staggers and Norcio (1993), but the content was initially proposed by Norman (1983). However, it is noteworthy that some definitions are still overlapping which does not resolve the problem of discerning the terms.

The *target system* refers to the interactive device that comprises a desired functionality for the user. The user is expected to learn how to operate and control that interactive device. At our study, the target system is denoted by 'the multimodal interaction style for music programming'.

The *conceptual model* refers to a representation of the target system that is devised by the designer or experimenter. The conceptual model should be accurate, complete, and consistent in helping the user to learn how to operate the target system. Metaphorical instruction is a form of conveying a conceptual model. In this study, the conceptual model is denoted by 'the roller metaphor'.

The *system image* refers to all devices and materials that make up the target system. In this study, the system image consists of the graphical, the sound, and interaction design, the physical devices, the content of the music collection, and the experimental instructions and setting.

The *mental model* refers to the internal representation of the target system that users create when they interact with the device. The mental model helps them to comprehend the system. It also contains the cognitive processes while a user completes a task with the target system. In this study, it is really an open question what users make up of the target system.

The *cognitive model* refers to a framework for conceptualizing user behaviour and knowledge acquisition. This framework includes the user's cognitive processes for memory, learning, attention, reasoning, etc. An experimenter attempts to construct a cognitive model by observing user behaviour. In this study, as part of a cognitive model, postulations are made how users think and act, when they interact visually or when they interact *non*visually with the same interactive device.

The *user model* refers to the representation of the user as it is programmed within the target system. It consists of elementary perceptual and cognitive abilities of users, or it takes account of preferences or custom settings of users. As such, the user model is, for obvious reasons, biased by the user impressions and user expectancies of the designer of the target system. A user model aims at improving the interaction between the user and the target system, and is used, for instance, in adaptive system and natural dialogue systems. In this study, the music recommendation

functionality contains, indeed implicitly, a user model, because it has made some assumptions how users select music.

The existence of a mental model is often inferred from problem-solving behaviour between novices and experts in domains of physics such as motion in space, motion of liquids, and mechanics (Gentner and Stevens, 1983). Mental model research in user-system interaction however mainly focuses on performance differences between users who are given, or are not given, a conceptual model of some interactive device. In this respect, giving a conceptual model to users can be interpreted as a form of providing foreknowledge to users; knowledge which refers to specific knowledge elements concerning a specific application (Freudenthal, 1998). For the sake of convenience, it is implicitly assumed that the provision of a conceptual model induces an appropriate mental model at the user. Assuming the existence of a mental model provides then explanations for different observed user performances. In this study, it is investigated what mental model develops without the provision of a conceptual model.

In general, it appears that a conceptual model improves user performances in task completions. Starting with a conceptual model does not only help users in executing procedures for completing tasks, but also helps them recover from errors, or learn novel tasks (Kieras and Bovair, 1984). Metaphorical instruction about a command-driven device appears to provide improved retention of the command language and to promote learning of distinctive procedures of command use (Payne, 1988). Metaphorical and procedural instructions on an interactive device allow both the young and elderly to perform better. No performance differences between the young and elderly exist, when they are both devoid of such foreknowledge. Hence, it is suggested that lack of foreknowledge is a major cause for the elderly performing worse than the young in controlling interactive devices (Freudenthal, 1998). An internal representation developed at one system appears to partly determine how efficient a user performs at another system (Payne, Squibb, and Howes, 1990). However, as the mental model of a user is not directly observable, the precise status of mental models with respect to their construction, modification, and content is still unclear. In this study, two methods were employed to infer aspects of a mental model. The first method was to have participants produce a post-task drawing of the interaction style for revealing relevant interaction aspects. The second method was to have participants choose a structure diagram representing the declarative organization underlying the interaction style.

10.2.2 Mental model in the absence of visual information

The visual conceptual model of the interaction style consists of a roller metaphor. Visual exposure to and working with this conceptual model can be interpreted as a form of foreknowledge for working with the same interaction style without visual display of information afterwards. In the case when this form of foreknowledge is not provided, users who interact nonvisually have to develop their private conceptual model, i.e., mental model, of the interaction style. Nonvisual sensory feedback, i.e., tactual and auditory feedback, has to compensate for the lack of visual feedback to develop an internal representation.

The first developments of a theory how users interact without visual information are much inspired by work on navigation and locomotion without sight (Klatzky, Loomis,

and Golledge, 1997). The authors have a long-standing interest in the navigation abilities for the blind, especially in developing a prototype navigation aid for blind people.

Just as is emphasized in the case of locomotion without vision (Klatzky, Loomis, and Golledge, 1997), it is assumed here that non-visual interaction with a device is limited by the lack of a mental representation. Nonvisual interaction grows out of knowledge about the physical environment. Without a visual display of information, knowledge of the spatial layout of the objects of interest is considered minimal information to act or navigate purposefully. In this study, the objects of interest are the music programming concepts represented by rollers in the interaction style. While the spatial layout comes almost for free when displayed visually, the construction and modification of an internal representation when there is no visually displayed information is considered contingent on the actual course of interaction. Without visual cues a user must rather rely on a sort of *dead reckoning* process, in which each new state of the interaction is determined by knowing the starting state and the consequences of the action. As the consequences of the action and the current state of interaction cannot be visually observed, other sensory modalities fed by auditory and haptic display need to compensate for that in a stepwise fashion. In contrast with the dead reckoning style of nonvisual interaction, a global visual perspective on the domain of interest facilitates planning ahead of actions. Visually represented objects can act as landmarks allowing the user to globally delineate the task along these landmarks. As the objects of interest can be literally pointed at as landmarks, actions can be determined beforehand.

The lack of visual display of information produces more uncertainty in the consequences of an action, and, consequently, in the current state of the interaction. In this respect, uncertainty means the difficulty of anticipating what a consequence of an action will be. Coping with these uncertainties will increase explorative behaviour with more cognitive effort to align uncertain outcomes with intended purposes. Tactual feedback is only present by performing an action, and is by definition absent when no actions take place.

Users, especially those who interact with a device without visual display of information, have to develop spatial knowledge, i.e., a cognitive map, of the environment in which they interact (see, e.g., (Evans and Pezdek, 1980; Thorndyke and Hayes-Roth, 1982; Presson, 1987; Sholl, 1996) for research on the development and representation of cognitive maps). Explorative behaviour enables the awareness of the objects of interest and their spatial structure, and the acquisition of knowledge about the actions that manipulates these objects and routes them across these objects.

Summarizing, the critical factor between visual and nonvisual, i.e., tactual and auditory, interaction is human memory. Within the context of our interaction style, the consequences of tactual and auditory feedback are momentarily and behave transiently, whereas the consequences of visual feedback are continuous and persist. Hence, it is expected that nonvisual interaction is less efficient than visual interaction, while holding the level of auditory and tactual feedback constant. Nonvisual interaction is expected to consume extra time between actions, which might be ascribed to additional cognitive processing for developing a cognitive map of the interaction style. Also, nonvisual interaction is expected to require a larger number of actions to complete a given task, which might be ascribed to explorative behaviour for developing the cognitive map, for explicitly inspecting the state of the interaction, and for monitoring the

progress of the task. However, as nonvisual interaction forces users to explicitly reveal and memorize the actions for navigation and manipulation, they ultimately develop a substantial body of procedural knowledge. It is hence expected that the score on procedural knowledge after nonvisual interaction is higher than after visual interaction.

Prior visual exposure to the interaction style is expected to facilitate nonvisual interaction afterwards. It is therefore assumed that users adopt a default conceptual model of an interactive device as a result of the visual exposure. This imagined representation is then used to retrieve information and to plan actions. More specifically, if users have formed a cognitive map that is aligned with the default conceptual model, it is expected that they benefit from this knowledge during nonvisual interaction with the same device; it is expected that they will perform more efficiently. As it assumed that users internalize and apply the default conceptual model as form of foreknowledge after visual exposure, it is expected that they use this foreknowledge to explain the interaction style to others. More specifically, it is expected that they focus their explanation on aspects pertaining to the visual representation of the interaction style. In contrast, users who are refrained from this foreknowledge need to follow other ways to explain the interaction style. Probably, they will devise their private metaphors or analogies for explanation. As nonvisual interaction requires an explicit account of actions and their consequences, it is expected that users tend to explain the interaction style by referring to actions and their nonvisual sensory effects.

10.2.3 Ease-of-learning and ease-of-remembering

Usually, first-time users of consumer electronics attempt to operate the device immediately without the aid of instructions. The application area of consumer electronics differs from the professional area with respect to training and instruction (Eggen, Westerink, and Haakma, 1996). Professional users are able to attend instructional and training courses and can count on continuing coaching and education for a specific complex device, whereas users of consumer electronics are offered no, or are simply not willing to take, opportunities for formal training. Additionally, reading the instructions or manual accompanying the product is often perceived too time consuming or too effortful. In some occasions, the instruction manual is simply lost. Hence, *learnability* (or ease-of-learning) is considered a fundamental usability criterion (Nielsen, 1993); the user should perceive at a glance the most efficient and effective ways to work with the product. This is certainly true for the domain of consumer electronics. It is of great concern how complex consumer electronic devices can be learned to operate with only a minimal need of instruction at the outset. Learnability is measured by the time or effort it takes to let users attain a specified level of proficiency.

Users of consumer electronic devices can be characterized as being casual in their interaction behaviour. They are not expected to use a given device with a great deal of systematicity, and they are not expected to become an expert in using all kind of device features. Brief, off and on interactions addressing a particular interest are considered stereotypical in this domain. *Memorability* (or ease-of-remembering) is another important usability criterion (Nielsen, 1993), especially for the domain of consumer electronics. A user must be able to return to an interactive device after a period of not using it, without the burden of re-learning it. Memorability is to a great extent determined by learnability, but facing an interactive system for the first time is definitely different from

returning to it. First-time learning often requires outside instruction or the experience of an illuminating moment in which the conceptual model is grasped. Once one realizes the concepts of the interaction style, the interactive device becomes rather memorable to use, or one may even be able to find novel procedures. A straightforward method to measure memorability is by measuring the time or effort it takes to let users, who have not used the interaction style for some pre-defined period, attain a specified level of proficiency.

10.2.4 Skill acquisition

Learning to operate a complex device might partly be interpreted as a rote memorization task; action sequences are simply memorized without attention for meaning. However, this type of learning is considered inadequate to deal with complex tasks and the acquired skill is not expected to be transferable to other domains or conditions. Hence, rote memorization is not addressed here. As described in Section , it is assumed that users assimilate some form of internal representation which allows them to reason about the device.

Cognitive, motor, and perceptual skills improve by practice. Hence, the qualitative nature of interactive control also changes while practising. Three stages of skill acquisition are proposed (Rasmussen, 1986).

> In early practice, control is more qualitative; attention is devoted to selecting the appropriate strategies for executing component tasks. Interactive control can then be interpreted as a problem solving task for selecting the appropriate strategies.
>
> Later on, the selection of the strategies themselves have become routine which leaves more attention span for the coordination of the actions within each strategy. Although the problem solving aspect has become less, all actions that are required to complete a task still need attentive behaviour to be performed smoothly.
>
> At a last stage, automaticity is expected to come into play in which the physical actions seem to be triggered by themselves with no cognitive load. One has done the task so often, that one does no longer need to concentrate on the task while doing it. It is assumed that one may even carry out other tasks at the same time.

The experiment deals only with learning at the early stages of product use. Because routine behaviour occurs at an expertise level, the last stage in skill acquisition is not considered here.

10.2.5 Declarative and procedural knowledge

At least two types of knowledge are relevant for learning to use an interaction style: *declarative* knowledge and *procedural* knowledge (Anderson, 1993). Declarative knowledge refers to knowledge of knowing *that*, in the sense that we know domain objects and relations between them. Procedural knowledge refers to knowledge of knowing *how*, in the sense that we know what actions are required to attain a specific goal.

The acquisition of one form of knowledge does not necessarily directly follow from having the other form of knowledge at one's disposal. In general, it is assumed that the objective of a task is well-known to the user, irrespective of whether this objective is an

outside instruction or is constituted at will. Additionally, users presumably possess knowledge of the task domain, irrespective of whether it is learned by instructions or learned by previous experience. However, novices encountering a new interaction style often find difficulties to transfer their declarative domain knowledge and task objectives to procedural knowledge on *how* to achieve the desired result. They are unfamiliar with what actions or what procedure will realizes the task objective. As an interaction style comprises the procedures of many tasks, the task space is perceived as infinitely large. Users have to solve the problem of finding effective means to attain a desirable end (Newell and Simon, 1972). However, in the initial stage of learning to operate an interactive device, it is claimed that the emphasis lies on the discovery of means, i.e., actions, instead of finding strategies for using these means. Likewise, Kosotsky and Simon (1990) claimed that the discovery of effective actions makes problems 'really hard', though their focus was on puzzles. In the case of interactive devices, users simply do not know or recognize what terminals (e.g. buttons, labels, switches, and toggles) are relevant to their current task without explicit instruction. Hence, the initial problem that first-time users have to overcome is to find out what exactly constitutes an action, what consequences can be expected from an action, and whether this result is effective with respect to their task objective. If the concept of an action and result, i.e., the 'perceived affordance' of an action, is problematic to perceive instantly, the discovery of purposeful actions induces a problem space and an associated cognitive load by itself.

In this experiment, users are informed about the music programming domain in a declarative way, while being left uninformed about the existing procedures, i.e., no procedural knowledge. The question is now to what extent users are able to perform tasks, i.e., procedures, to what extent users are able to acquire procedural knowledge, and to what extent users are able to acquire declarative knowledge.

10.3 Hypotheses

The following hypotheses are defined:

(i) visual interaction is more efficient than nonvisual interaction, while leaving the level of auditory and tactual feedback in both conditions constant;
(ii) users who have *no* visual display, while leaving the level of auditory and tactual feedback in both conditions constant, have a higher score on procedural knowledge than users who have a visual display;
(iii) users who are visually exposed to and worked with a visual display and are subsequently transferred to a condition without a visual display perform more efficiently than users who start working without a visual display;
(iv) users who have a visual display make a re-production of the interaction style that contains more visual aspects, whereas users who have *no* visual display make a re-production which contains more action-and-effect related aspects.

10.4 Measures

A measure of task performance is required to test Hypotheses (i) and (iii). Task performance is described in Section 10.4.1. A measure of transfer is required to test Hypothesis (iii) and is described in Section 10.4.2. The score of procedural knowledge is assessed by

a post-test questionnaire containing questions about small interactive procedures with the interaction style. It is required for testing Hypothesis (ii) and it is described in Section 10.4.3. A device to measure the content of a user's internal representation of the interaction style is described in Section 10.4.4. It is used for testing Hypothesis (iv).

10.4.1 Task performance

In order to measure task performance, two measures, *number of actions* and *compilation time* required to perform the task, are defined. All actions on the trackball (e.g., roll movements, presses) are logged with a time indication, and written to a file for later analysis. Number of actions and compilation time are measured between the first action and the last action of the compilation process.

10.4.2 Transfer

In general, transfer is a measure of increased performance of an experimental group, who experiences a change in experimental conditions, relative to another group who receives no change (Woodworth and Schlosberg, 1965). For our experiment, it is assumed that all tasks in the experiment are alike in difficulty. The transfer for this experiment is defined as the difference of task performance scores between participants who perform tasks with a particular interaction style after they have done tasks using a first interaction style, and participants who start performing tasks using that particular interaction style.

10.4.3 Procedural knowledge

Procedural knowledge is measured by means of two 20-item questionnaires handed out half-way during the experiment, and at the end of the experiment. The two questionnaires contained a relatively large overlap of questions (16) as well as four distinct questions. Half of the questions in each questionnaire represented questions on procedures consisting of a single action (single step interaction). The other half represented questions on procedures that could be performed by 2 or 3 actions (multiple step interaction). The order of questions in both questionnaires was randomized. The order in which the two questionnaires were given was counterbalanced. This questionnaire design allows measurement of transfer, minimizes item-learning, and treats all sensible questions about procedures that could be constructed. Participants answered the questions in verbal form. The test supervisor coded the answer, making use of a short-hand notation, elaborated in Appendix II. Answers to items could not be withdrawn or corrected when concluded.

10.4.4 Mental model

Two methods were employed to infer some aspects of a mental model. The first method was to have participants produce a *drawing* to reveal what aspects of the interaction style are relevant. To derive the declarative content of the mental model, the second method provided participants with *structure diagrams* to choose from. For experimental

convenience, participants chose a diagram from a set of alternatives, rather than make a diagram themselves.

10.4.4.1 Drawing

Participants were asked to imagine a situation in which they had to explain the interaction style to a person, who is unfamiliar with the interaction style (teach-back method (Van der Veer, 1994). To aid the explanation, they were instructed to produce a free drawing of the interaction style, containing all necessary details to explain the interaction style. The test supervisor did not intervene while participant drew the interaction style. After the drawing was completed, the participants were asked to explain their picture to the test supervisor. This explanation was primarily for the experimenter to interpret the drawing correctly.

10.4.4.2 Structure diagram

The declarative knowledge of the interaction style contains the particular organizational structure of the music collection. By letting participants choose a structure diagram, which represents different ways of how the music collection might be organized, a simple method is devised that indicates how accurate the declarative knowledge component of their mental model is. The graphical notation of the structure diagram was explained to the participants (see Figure 28).

The diagrams picture three alternative hierarchical relations between the concepts 'collection', 'styles', 'tracks', and 'recommendations', in that order. Each relation between these concepts can, in principle, be arranged in two different ways. Thus, in total, eight structure diagrams are composed. Only one gave an accurate representation of the music collection. Each structure diagram was assigned to a category, denoted by integers from 0 to 3; category 0 represents the correct diagram, categories 1 to 3 reflect the alternative diagrams that have a corresponding number of wrong relations (see Figure 28).

If we compare the correct diagram and the most incorrect diagram (assigned to category 3), we observe three differences in the relations between the concepts. First, the incorrect diagram lacks a music style that contains all other music styles (the upper oval in the correct diagram). Second, the incorrect diagram makes it appear that pieces of music belong to more than one music style. Third, the incorrect diagram makes it appear that music recommendations of a particular piece of music belong to different music styles.

The structure diagrams were also shown half-way during the experiment and at the end of the experiment. Each hand-drawn diagram was displayed on a separate A4 format sheet of paper. The sheets of papers were labelled and laid out on the desk top. The layout on the desk top was different when they were presented to the participants for the second time. Participants were asked to choose the structure diagram from the eight alternatives, as shown in Figure 28, that most closely matched the underlying structure of the music collection.

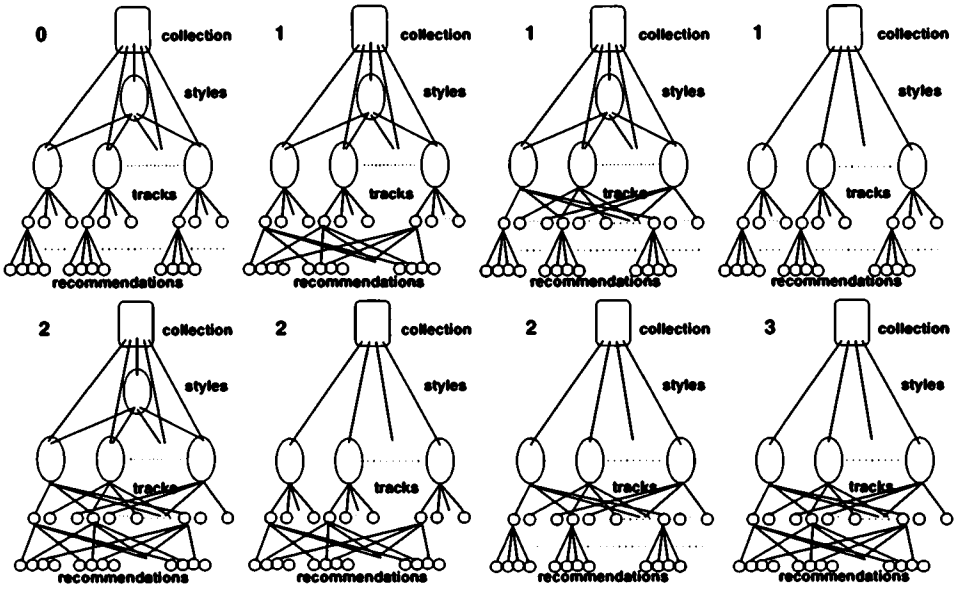

Figure 28. Eight structure diagrams that each might represent the organizational structure of the music collection. The scores are shown in the left-upper corner of each diagram, score 0 is reserved for the correct one.

10.5 Method

10.5.1 Pre-test

In a pilot experiment, it appeared that participants needed additional training with the IPO force-feedback trackball (Engel, Haakma, and van Itegem, 1990). When using the IPO force feedback trackball for the first time, participants were unaware what basic actions are possible and what tactual effect these actions induce. In addition, they were unfamiliar how these actions are expected to be performed by what hand-finger movement configuration, by what force, and by what level of accuracy. Second, the perception of force feedback is an experience that has to be explicitly learnt to be felt. Third, as participants are unfamiliar with the basic actions of the trackball, they were simply not able to perform complex tasks with it. So, participants were given a short introduction on pressing and double-pressing the trackball. In the form of a pre-test, participants were acquainted with the four main directions in which the trackball could be rolled while experiencing force feedback. This type of roll movements was also present in the interaction style. The tactual objects, i.e., force fields, were parameterized in the same way as in the interaction style.

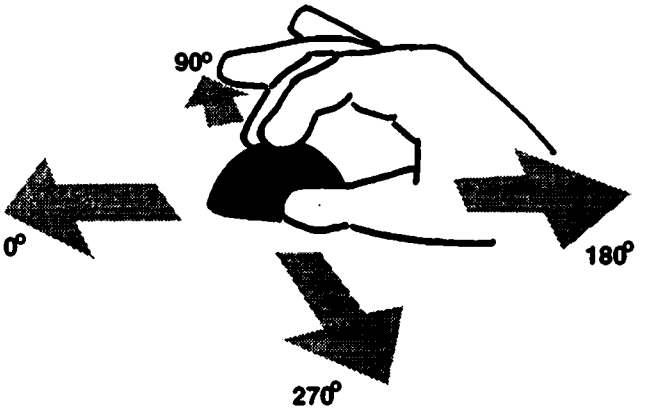

Figure 29. The four movement displacements measured in degrees correspond to a right hand. The index and middle finger are considered most comfortable for controlling the ball.

10.5.1.1 Procedure

The pre-test consisted of four movement displacement conditions: the trackball had to be rotated 1/16th part of its circumference (approx. 1.5 cm at the surface) in four directions: forward at 0°, to the left at 90°, backward at 180°, and to the right at 270°. The movement displacements for a right-handed participant are shown in Figure 29. Using a two-alternative forced choice method (2AFC), a reference force or no force was presented in a random order while the participant was asked to rotate the ball in a predefined direction. For all four directions, the reference force was held fixed as determined by a constant level of motor currents (i.e., set points). The fixed set point value of 400 equalled an actual force value of 86 grams (Keyson, 1996). The four movement displacement were also randomized over 60 trials. No practice trials were conducted.

The participants were seated in a non-reverberant room in a comfortable chair, in front of a monitor. The IPO trackball was placed at a small table next to the chair, in such a way that the hand of the participant managed to control the trackball in a comfortable way. The participants were instructed to place both the index and the middle finger in parallel near the ball's top, but just above its surface, at the outset of each movement. In previous experiences, it was found that a configuration in which both the index and middle finger are positioned on the trackball, as shown in Figure 29, provides a high level of control.

Each trial consisted of two subsequent actions, one of which was accompanied by force feedback. A 80 ms 900 Hz tone was sounded to indicate the start of the movement in which either no force or the reference force was applied randomly. The participants were instructed to rotate the ball in a smooth movement. When correct, a 80 ms 1300 Hz tone was sounded. If they rotated the ball in an inappropriate direction, three 70 ms 300 Hz tones spaced 70 ms apart marked the erroneous movement; participants were then allowed to repeat the movement. The participants were instructed to release the ball after hearing the tones to prepare themselves for the next movement, by re-positioning

their index and middle finger. The second action proceeded identically, but was preceded and completed by two tones, rather than one. Participants then indicated in which movement they had experienced a tactual force that was described as 'feeling a small rib'. After that, the words 'Right' or 'Wrong' appeared on the screen as feedback.

10.5.1.2 Results

The mean percentage correct responses per movement direction are shown in Figure 30. As might be expected, the data show ceiling effects; the correct percentages closely approach 100%. A logit transform of the proportion correct, while compensating for ceiling effects by Bayes' correction, is used in the analysis. More specifically,

$$\text{logit}(p) = \log\left(\frac{\frac{C+1}{N+2}}{\frac{N-C+1}{N+2}}\right)$$

where p is the proportion correct, C denotes the number of correct responses, and N denotes the total number of responses.

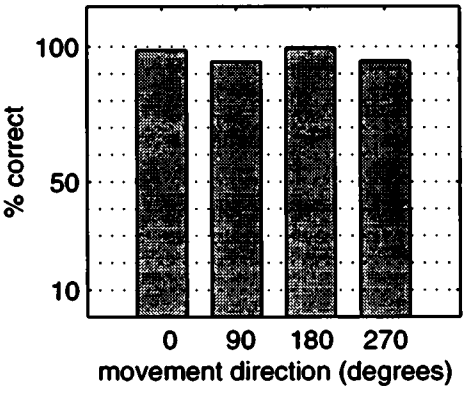

Figure 30. Mean percentage correct responses of 2AFC experiment for discriminating tactual force feedback in trackball movements.

An ANOVA analysis with repeated measured in which *movement direction* was treated as a within-subject independent variable and the transformed proportion correct as dependent variable revealed a significant *movement direction* effect ($F(3,69) = 7.674$, $p < 0.001$). Examination of the results, as shown in Figure 30, reveals that participants were better in perceiving tactual cues when moving the index and middle finger forward or backward on the ball (at 0° and 180°) than moving laterally (at 90° and 270°). In general, participants could accurately perceive tactual cues in all directions as demonstrated by the high number of correct responses; the size of *movement direction* effect is only small.

10.5.2 Music programming experiment

10.5.2.1 Instruction

In order to provide declarative knowledge, participants were informed about the concepts in the domain. In the music selection and programming domain, participants have to comprehend domain objects such as the music collection, the music styles, the pieces of music, and the music programme, the interrelationships between these objects, and the conceptual actions that can be invoked on the domain to alter object states or relations between objects.

Participants read a short text about the music programming domain. They were left uninformed *how to* manipulate the controls or *how to* navigate through the music collection, i.e., the procedural knowledge. A teach-back method (to the test supervisor) was used to guarantee that the concepts were comprehended as intended; participants were asked to rephrase the given short text in own words; any misconception of the music programming domain was corrected by the test supervisor. The precise instruction text is presented in Appendix I. In addition, participants received a text explaining the musical characteristics of the jazz styles in the music collection. Each explanation of a jazz style referred to a small number of typical jazz musicians who had made contributions to the development of the jazz style.

At the outset of each music programming task, participants received a written task description. Participants were instructed to compile a music programme as quickly as possible while paying no attention to their personal preferences or the order of pieces in the resulting programme. The task was ended when the 10 firstly compiled pieces of music were all distinct and were equally drawn from two pre-defined jazz music styles. The precise task instruction text is presented in Appendix I.

10.5.2.2 Design

Four condition were applied that are shown in Table 1. In one control condition, denoted by VAT (Visual, Auditory, Tactual feedback), participants completed four tasks by using the complete multimodal interaction style, denoted by O. The four consecutive tasks are denoted by *task number*. In the other control condition, denoted by AT (Auditory, Tactual feedback), participants only worked with the interaction style without visual display, denoted by X, for all four tasks. At interaction style X, the display monitor was physically removed. In the other two experimental conditions, participants worked with both interaction styles, one after the other. The condition, denoted by VAT->AT, there was no visual display for the last two tasks. In the last condition, denoted by AT->VAT, this was reversed.

10.5.2.3 Test material and equipment

A music collection comprising 480 first-minute excerpts of jazz music pieces (MPEG-1 Layer II 128 Kbps stereo) from 160 albums served as test material. The collection was assembled by considering 12 jazz styles. These styles cover a considerable part of the whole jazz period. Each style contained 40 pieces of music. The main test equipment

Table 1. Design of the experiment. VAT and AT refer to control conditions in which participants worked respectively with and without a visual display. VAT->AT and AT->VAT refer to experimental conditions in which participants both worked with and without a visual display, one after the other. O and X respectively refer to the interaction styles with a visual display and without a visual display.

	task number			
condition	**1**	**2**	**3**	**4**
VAT	O	O	O	O
AT	X	X	X	X
VAT->AT	O	O	X	X
AT->VAT	X	X	O	O

consisted of a Dell GXPRO-180, running under Window 95, on which the interaction style was implemented. All music encoded as MPEG data was stored on the hard disc. Real-time MPEG decoding was done by software. Digital audio was converted to analog by a Soundblaster 16 audiocard. A second PC, a Dell 466/Le, was used for controlling the IPO trackball. Two I/O cards, a digital-to-analog (DAC) and an analog-to-digital converter (ADC) enabled communication between the PC and trackball. Both computers communicated by the standard parallel communication ports (RS-232). The audio was amplified by an audio amplifier (Philips DFA888) through a pair of high-quality loudspeakers (Philips 9818 multi-linear 4-way).

Participants were seated in front of a 17-inch monitor (Philips Brilliance 17A), if present, in a non-reverberant studio, originally designed for speech recordings. They were seated in a comfortable chair. They could adjust the audio volume to a preferred level. The IPO trackball was placed on a small table next to the chair, in such a way that the hand of the participant managed to control the trackball in a comfortable way.

10.5.2.4 Procedure

Participants performed two experimental sessions on two separate days. They were randomly assigned to one of the four conditions. Of the 24 participants, 17 returned on two consecutive days, five participants had one day in between both sessions, one participant had a weekend in between the sessions, and one participant returned 6 days later for the second session. The first session started with an intake questionnaire for obtaining mainly personal data and previous experiences with input devices. Subsequently, participants were accustomed to the required motor skills to control the IPO trackball with force feedback in a familiarization phase, i.e., the pre-test. This phase took approximately 15 minutes.

After this phase, participants were informed about the music programming domain by reading the instruction. Participants 'taught back' the content of the text to the test supervisor by explaining the text in their own words; any misconception was repaired immediately. Next, participants could freely explore the interaction style (with or with-

out a visual display) for three minutes. The second session started immediately with a 3-minute exploration phase.

At the outset of each individual task, participants received a written task description. Again, participants were asked to rephrase the task instruction to avoid any misconceptions of the task.

The participants completed two music programming tasks during each session. Subsequently, participants were brought from the place of operating the interaction style to a desk from which it was impossible to view the test equipment. First, they were given the opportunity to make remarks on or to criticize the interaction style. Second, the procedural knowledge questionnaire was completed in a dialogue with the test supervisor. Third, they were asked to draw and explain afterwards a sketch of the interaction style, that would help in explaining the interaction style to a person who is unfamiliar with the interaction style. Finally, participants were asked to choose the structure diagram that most closely matched the inner organization of the music collection.

10.5.2.5 Tasks

Four music programming tasks were defined and counterbalanced. The tasks were designed to be equally difficult; a successful and most efficient task completion demanded 23 actions. The precise instruction of the tasks are presented in Appendix I.

The number of 23 actions can be clarified by considering the interaction style in more detail (see Figure 27). The most efficient task execution accesses only the two rollers in the middle: the music styles and the music collection roller. Each task starts at the music styles roller. In addition, the music has to be selected that appears as first on the roller; so, no searching for preferred music is allowed for efficiency reasons. The two music styles, though all music styles differ in the four tasks, and the initial state of the music styles roller were chosen in such a way that they were all ideally a single action apart from each other. The first half of the most efficient task action sequence, now, consists of one action to select a music style, an action to go to the music collection roller, five actions to add music to the programme, alternated by four actions to go to the next piece of music, and an action to go back to the music styles roller. This concludes the first half of music programming task which took 12 actions. The second half comprises 11 actions starts again with an action to select a music style, an action to go to the music collection roller, and 9 actions to compile music. Summing the two halves amounts to 23 actions.

10.5.2.6 Participants

Half of the 24 participants (18 male, 6 female) were recruited by advertisements and all got a fixed fee. The other half consisted of colleague researchers working at the IPO. Eight persons had already participated in former experiments on music programming. The average age of the participants was 28 years (min: 21, max: 45). Participants were not selected based on their musical preferences or musical education. All participants had at least a higher level of vocational education. Two persons (not included in the total number of 24 participants) experienced difficulties at the first compilation task. One person did not acquire the required proficiency to control the interaction style after

the 3-minute exploration phase. Another person had interpreted the exploration phase as a music listening phase and 'forgot' to discover facts about the interaction style; the test supervisor had to instruct the participant on procedures during the first compilation task. Data of both participants were excluded from the analyses; only data of the remaining 22 participants were analysed.

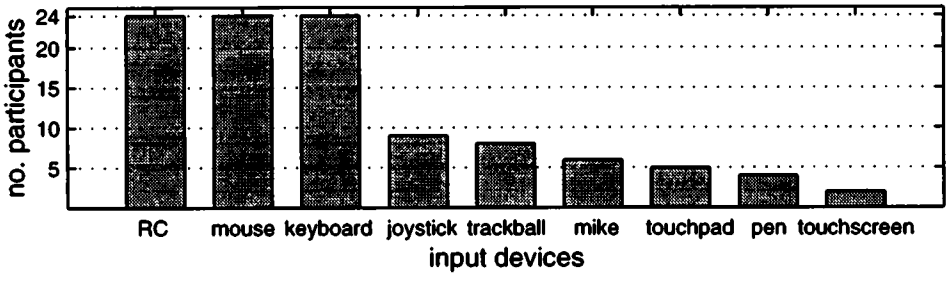

Figure 31. The number of participants that responded to be familiar with a particular input device.

Participants were asked on the basis of a list with which input devices they had worked before. All participants were familiar with using a remote control (e.g., for television sets), mouse, or keyboard as shown in Figure 31. Eight participants had some experience in using a trackball; two of them had worked with the IPO force feedback trackball before.

Five participants responded they possessed a collection exceeding 100 CDs; the other 19 participants had a music collection containing less than 100 CDs.

With respect to the frequency of CD programming facilities use on current CD players, it appears that this functionality was rarely used. Twenty participants responded that they never[1] or seldom used the programming functionality. If used, it was mainly for making recordings on other media or for skipping non-favourite tracks. In general, participants thought it took too much effort and was too time-consuming to program or to learn to program a CD; they tend to play back a whole CD from its beginning until its end, or just select one particular track at a time.

Of the 24 participants, two responded that they always played music in a random order. The other group consists of 17 participants who never or seldom use the functionality, and five participants who sometimes or frequently use the functionality. If used, it was mainly for the surprise and variation effects. In general, participants responded they rather liked listening to the music in the original order. Five participants simply did not feel the urge to use it or never thought about it.

1. The questions on frequency of CD functionality use were posed by using the ordinally ordered responses: never, seldom, sometimes, frequent, often, and always.

10.6 Results

This section describes the analysis results of the experiment. The first analysis concerns the investigation of global features, trends, and extreme cases in the raw data: compilation time and number of actions. Descriptive analysis for investigating trends and extreme cases is elaborated in Appendix III.

One participant's behaviour was suspected to consist of music exploration based on the fact that he spend more than three times as much time and actions than the average on three compilation tasks. In addition, the participant freely admitted that he had searched for preferred music for the first three tasks. As the task objective was to perform the task as quickly as possible without taking care of personal music preference, the data from this participant are excluded from the analyses concerning the compilation time and selection effort. For the other analyses, the data of this participant were considered still valid and were left unchanged.

Despite the short introduction on pressing the trackball, double-clicking the ball was considered problematic for six participants. This problem is a shortcoming of the implementation; the trackball should not be moved while double-clicking it. If, during a double-click, the trackball is only slightly rotated, two single clicks are registered. The logging data was adjusted by replacing each sequence of multiple single clicks by a single execution of a single click. Also, the time associated with these sequences was removed from the data. This adjustment resulted in fewer actions and lower compilation time, but had no consequences on the results of the analysis.

In the following two Sections 10.6.1 and 10.6.2, the data of compilation time and number of actions over the different experimental conditions are analysed. Subsequently, the completed questionnaires assessing procedural knowledge are analysed in Section 10.6.3. The drawings made by participants are analysed in Section 10.6.4. The choices for structure diagram are analysed in the concluding Section 10.6.5. The spontaneous remarks of the participants are reported in Section 10.6.6.

10.6.1 Compilation time

Compilation time is defined as the time elapsed between the instants of the first action and the last action that completed the task. The cell means and the marginal means for the compilation time are shown in Table 2. A graphical representation of the mean compilation time of the tasks in both the experimental and the control conditions is shown in Figure 32.

An ANOVA with repeated measures was used with *task number* as within-subject independent variable and the four *conditions* (two experimental and two control conditions) as a between-subject independent variable, and compilation time as dependent variable. A significant *task number* main effect ($F(3,57) = 3.642$, $p < .05$) was found. Also, a significant *task* by *condition* interaction effect ($F(9,57) = 3.257$, $p < .005$) was found. With respect to the *task* main effect, the linear trend in the data was significant ($F(1,19) = 5.965$, $p < .05$) in a post-hoc polynomial contrast analysis. Inspecting the *task* marginal means in Table 2, it can be concluded that the compilation time has decreases with *task number*. Thus, participants needed less time to compile each successive music pro-

gramme. With respect to the *task* by *condition* interaction effect, a post-hoc contrast analysis revealed that the first task in the AT->VAT condition had a significantly higher compilation time than the first tasks in other conditions ($F(3,19) = 4.168$, $p < .05$).

Table 2. Mean time to compile a music programme containing 10 pieces of music (seconds). Marginal means are shown in the right column and bottom row.

	task number				
condition	**1**	**2**	**3**	**4**	
VAT	86.8	73.8	88.8	73.0	80.6
AT	184.5	168.5	115.5	145.3	153.4
VAT->AT	102.8	113.6	139.4	110.4	116.5
AT->VAT	202.6	139.3	96.1	79.9	129.5
	144.6	124.0	113.3	100.7	120.6

To investigate the possible transfer effect caused by the visual display, the performance of participants who worked without a visual display after they had done two tasks with a visual display, i.e., the last two tasks in the VAT->AT condition, is compared with the performance of participants who worked without a visual display in the first two tasks in the AT->VAT condition. Only data from the experimental conditions was used in order to keep the number of participants equal in both groups.

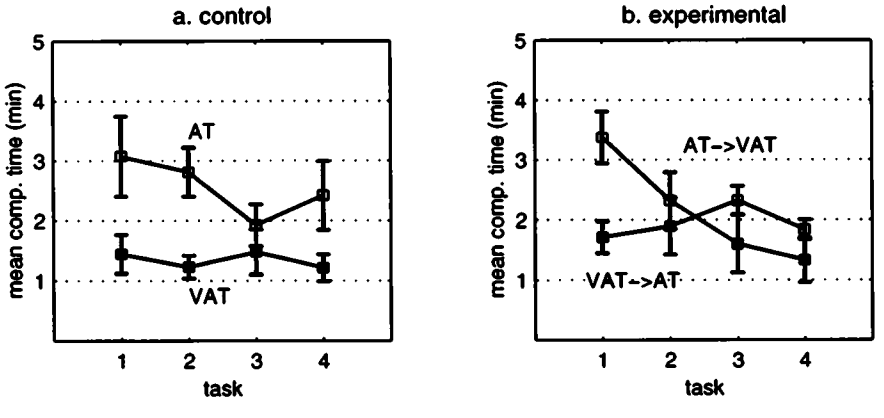

Figure 32. Mean compilation time (minutes). The left-hand panel (a) shows the mean compilation time for all four tasks of the two control conditions (AT and VAT). The right-hand panel (b) shows the mean compilation time for all four tasks of the two experimental conditions. The cross-bars represent standard error of the mean.

An ANOVA with repeated measures in which *transfer* was treated as a between-subject variable, and the two tasks as a within-subject variable was carried out. Compilation time was the dependent variable. A significant *task number* main effect ($F(1,13) = 13.179$, $p < 0.003$) was found. Participants needed less time to compile a programme for each successive task. Although two preceding tasks with a visual display decreased the mean compilation time by 46 seconds for participants who subsequently worked without a visual display, the transfer effect ($F(1,13) = 3.251$, $p = 0.095$) was not significant.

The ANOVA for investigating a possible transfer effect the other way around did not produce significant effects.

10.6.2 Number of actions

The cell means and the marginal means for the number of actions are shown in Table 3. A graphical representation of the mean number of actions for the tasks for both the control and experimental condition is shown in Figure 33.

An ANOVA with repeated measures was used with *task number* as within-subject independent variable and the four *conditions* (two experimental and two control conditions) as a between-subject independent variable, and *number of actions* as dependent variable. Only a significant *task number* main effect ($F(3,57) = 3.793$, $p < .05$) was found. In particular, a linear trend in the data was significant ($F(1,19) = 7.924$, $p < 0.05$) in a post-hoc polynomial contrast analysis. In general, participants performed fewer number of actions at each successive task, which can be at best seen in the marginal means for *task number* in Table 3.

Table 3. Mean number of actions to compile a music programme containing 10 pieces of music. The marginal means are shown in the right column and bottom row.

	task number				
condition	**1**	**2**	**3**	**4**	
VAT	53.3	38.8	50.3	33.5	43.9
AT	55.7	49.8	44.6	40.0	47.5
VAT->AT	49.1	50.9	44.3	37.6	45.5
AT->VAT	63.4	49.7	42.7	34.9	47.7
	55.7	49.8	44.6	40.0	47.5

To investigate the possible transfer effect caused by the visual display, the performance of participants who worked without a visual display after they had done two tasks with a visual display, i.e., the last two tasks in the VAT->AT condition, is compared with the performance of participants who worked without a visual display in the first two tasks

in the AT->VAT condition. Only data from the experimental conditions was used in order to keep the number of participants equal in both groups.

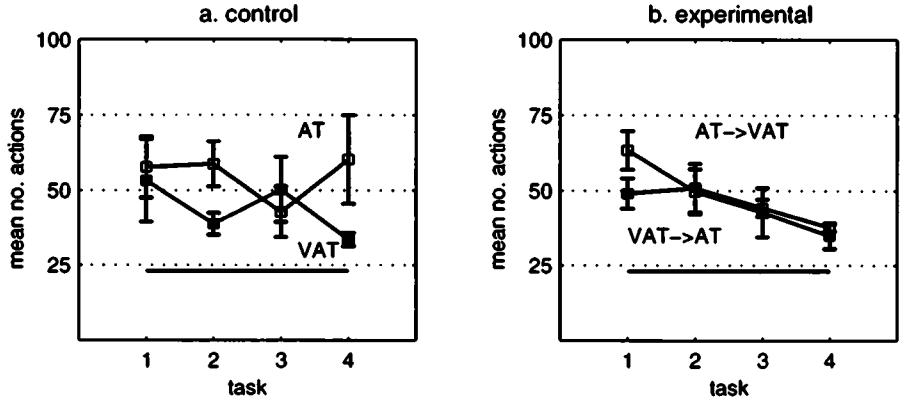

Figure 33. Mean number of actions executed. The left-hand panel (a) shows the mean number of actions for the four tasks in the control conditions. The right-hand panel (b) shows the mean number of actions for the four tasks in the experimental conditions. The minimal number of actions to complete the task is 23 (horizontal line). The cross-bars represent the standard error of the mean.

An ANOVA with repeated measures in which *transfer* was treated as a between-subject variable, and the two tasks as a within-subject variable was carried out. The number of actions was the dependent variable. A significant main effect of *task number* ($F(1,13) = 6.554$, $p < 0.05$) was found. Fewer actions were performed for each successive task carried out without a visual display. A significant *transfer* effect was found ($F(1,13) = 7.602$, $p < 0.05$). The performance of two preceding tasks with a visual display seems to significantly decrease the mean number of actions in two successive tasks without a visual display by 16 actions.

However, an improvement in performance between participants who carried out a third and fourth task and participants who carried out a first and second task can also be caused by mere practice. To investigate whether both a change in conditions and practice caused the performance improvement, the performance of participants at the last two tasks in the VAT->AT condition was compared with the performance of participants at the last two tasks in the AT condition. An ANOVA analysis reveal no significant main effect. The ANOVA analysis only revealed a significant *task number* by *condition* interaction effect ($F(1, 10) = 8.349$, $p < 0.05$); relatively more actions were needed in the fourth task in the AT condition.

The ANOVA for investigating a possible transfer effect the other way around did not produce significant effects.

10.6.3 Procedural knowledge

The level of procedural knowledge was assessed by a questionnaire that contained items asking what action sequence was required to fulfil a given sub-task. The items

were categorized into two equally sized groups: 10 questions concerning single step interactions and 10 questions concerning multiple step interactions. The number of correct answers in the questionnaire determines the questionnaire score. Answers were judged correct, if the responded action sequence actually fulfilled the sub-task (i.e., effective) and the action sequence was considered of minimal length (i.e., efficient). Answers were judged incorrect otherwise. All correct answers were summed to arrive at a questionnaire score. The questionnaires were completed at the end of the first and second experimental session. The mean questionnaire scores for the different conditions are shown in Table 4. A plot is given in Figure 34.

An ANOVA with repeated measures was used with *session* (2) and *kind of question (2)* as the within-subject independent variables, and *condition* (4) as the between-subject independent variable. The *condition* variable represents the two experimental and two control conditions. Questionnaire score was the dependent variable.

A significant main effect of *session* ($F(1,40) = 7.809$, $p < 0.05$) was found. Participants were better in completing the questionnaire for the second time. Also, a significant main effect of *kind of question* ($F(1,20) = 11.94$, $p < 0.005$) was found. Participants were better in answering the questions concerning the multiple step interactions than the question concerning the single step interactions.

The two-way *condition* by *kind of question* interaction effect ($F(3,20) = 2.974$, $p = 0.056$) was just not significant.

The three-way *session* by *condition* by *kind of question* interaction effect was significant ($F(3,20) = 4.803$, $p < 0.05$). As shown in Figure 34, participants who had worked without a visual display, i.e., the AT and AT->VAT conditions, were better in answering the multiple step interaction questions in the first questionnaire than the participants who had worked with a visual display, i.e., the VAT and VAT->AT conditions. In the second questionnaire, participants in the AT condition slightly improved in answering questions about multiple step interactions. Participants in the AT->VAT condition did not improve. On the other hand, participants in the VAT->AT condition improved gave more correct answers on multiple step questions the second time.

Table 4. Mean questionnaire score (single step interaction + multiple step interaction). The marginal means are shown in the right column and bottom row. The maximum score is 20.

	questionnaire score		
condition	**1**	**2**	
VAT	11.8 (6.0 + 5.8)	14.0 (7.2 + 6.8)	12.8 (6.6 + 6.2)
AT	14.0 (6.0 + 8.0)	15.3 (6.5 + 8.8)	14.7 (6.3 + 8.4)
VAT->AT	13.3 (6.5 + 6.8)	14.5 (6.4 + 8.1)	13.8 (6.4 + 7.4)
AT->VAT	14.1 (6.1 + 8.0)	16.0 (8.1 + 7.9)	15.06(7.1 + 7.9)
	13.4 (6.2 + 7.2)	15.0 (7.1 + 7.9)	14.3 (6.7 + 7.6)

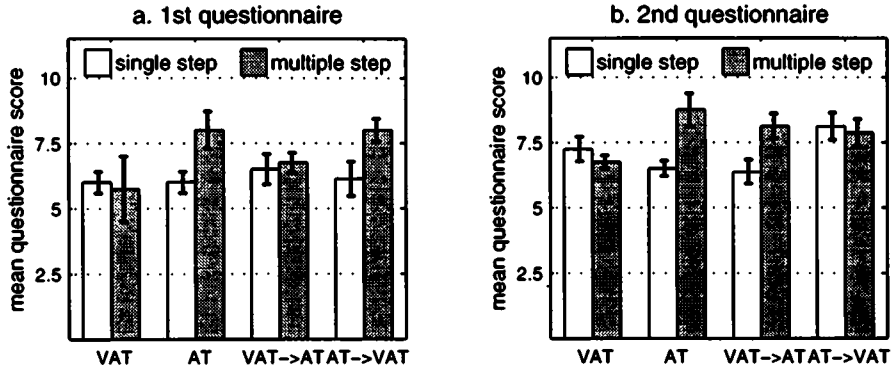

Figure 34. Mean questionnaire score. The left-hand panel (a) shows the mean questionnaire score for the control and experimental conditions at the end of the first experimental session. The right-hand panel (b) shows the mean questionnaire score at the end of the second experimental session. The white bars represent the mean question scores for the single step interactions. The grey bars represent the scores on the questions concerning the multiple step interactions. The vertical lines go through points a single standard error (SE mean) above and below the corresponding mean.

10.6.4 Drawings

At the end of each experimental session, participants were instructed to produce a drawing to explain the interaction style.

Four drawings and explanations are shown in Figure 35. Some participants who had worked without a visual display were rather creative in developing their own metaphor (see Drawing A and B in Figure 35). In general, drawings between participants who had worked with a visual display and participants who had worked without a visual display were qualitatively different. The drawings were analysed by judging whether typical aspects of the interaction style were present. This judgement was carried out after the experiment was concluded. The following aspects were counted:

1 Action-related aspects (e.g., rolling, pressing, and double-pressing the trackball);
2 Domain object-related aspects (e.g., 'collection', 'programme');
3 Force feedback-related aspects (e.g., drawing a hole or ripple);
4 Relationships between domain objects (e.g., by drawing a structure diagram);
5 Aspects related to physical devices (e.g., a trackball device, or a monitor);
6 Non-speech sound related aspects (e.g., 'hearing the sound of a creaking door');
7 Visual aspects pertaining on the roller metaphor (e.g., rollers or a graphical menu structure).

Parts of the results are shown in Figure 36; only aspects that were prominently present in the drawing (with a proportion higher than 0.5) are shown. The reason that force-feedback aspects are shown in Figure 36, despite its rare occurrence, is due to the fact that only some participants who had worked without a visual display drew force feed-

back-related aspects. Drawing force fields is not obvious. It appeared that drawing physical devices and relationships between domain objects rarely occurred. Both physical devices and the organizational structure of the music programming domain were considered irrelevant to explain the interaction style. Only one participant wrote a sentence about non-speech audio once. Again, drawing aspects related to non-speech audio is difficult.

As shown in Figure 36, participants in both control conditions produced two consistent drawings. This is however not true for the experimental conditions, in which participants received a change in conditions.

As shown in panel (a) of Figure 36, participants in the experimental conditions were rather inconsistent in drawing action-related aspects. After having worked without a visual display, more action-related aspects were drawn in the first drawing (6 out of 8) than in their second drawing (4 out of 8). In contrast, participants who had first worked with a visual display drew fewer action-related aspects in the first drawing (1 out of 8) than in the second drawing (3 out of 8). However, a test on independent proportions (McNemar, 1962; p. 52) showed that participants did not produce a change in drawing action-related aspects when experiencing a change in visual display conditions ($z = 0.633$). In the first drawing, participant who had worked without a visual display drew more action-related aspects (9 out of 12) than participants who had worked with a visual display (4 out of 12). A Fisher exact probability test (Siegel, 1956; p.96) just rejected equal proportions for both groups ($p = 0.0498$). Thus overall stated, participants who had worked without a visual display drew more action-related aspects.

As shown in panel (b) in Figure 36, participants in the experimental conditions tended to draw fewer domain objects in the second drawing than in the first drawing (8 out of 8 in the first drawing, 6 out of 8 in the second drawing). A test on difference between non-independent proportions showed that participants change their drawing of domain objects under a change of visual display conditions ($z = 3.464$, $p < 0.0005$).

As shown in panel (d) in Figure 36, all VAT->AT participants drew visual aspects in the first drawing. They however drew fewer visual aspects the second time (6 out of 8 in the second drawing). Again, AT->VAT participants drew fewer visual aspects in the first drawing (3 out of 8) than in the second drawing (6 out of 8). Surprisingly, two participants produced a roller metaphor themselves without ever having worked without a visual display. A test on the difference between non-independent proportions showed that producing visual aspects in drawings changed under a change in visual display conditions ($z = 2.111$, $p < 0.05$). In the first drawing, participants who had worked with a visual display drew more visual aspects (11 out of 12) than participants who had worked without a visual display (3 out of 12). A Fisher exact probability test rejected the hypothesis of equal proportions for both groups ($p = 0.00138$). Thus overall stated, participants who had worked without a visual display produced fewer visual aspects of the interaction style.

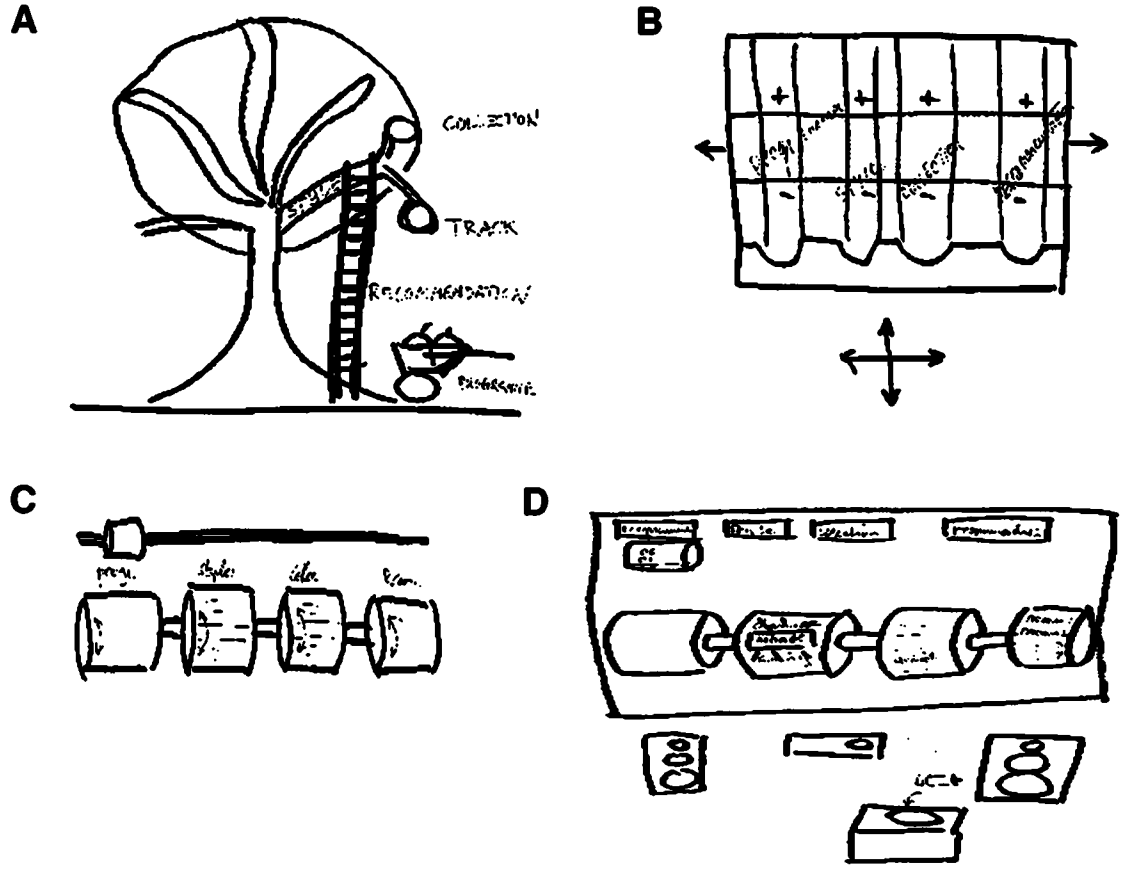

Figure 35. Four drawings made by different participants. Drawing A and B were made by participants, who had worked without a visual display. Drawing C and D were made by participants, who had worked with a visual display. Participant of drawing A explained the music collection as a fruit tree, in which the branches represents the music styles, and the fruits carried by these branches represents music tracks within a particular music style. Compiling a programme was clarified as filling a barrow with fruit; one can go along the hard way by climbing the tree along the trunk and reach for the fruit by shaking the branches, or one can decide to follow the easy path of recommendations by climbing a ladder that directly leads to the fruit of interest. Only domain objects and their interrelations were judged to be present. Participant of drawing B explained the interaction style as a coin collector. Each slot in the collector, which could clearly be felt as such, represents one of the concepts in the music programming domain. Being in one of these slot, one is able to inspect and feel each coin, i.e., piece of music, one-by-one. Only domain objects, actions, and force feedback were judged to be present. Participant of drawing C explained the roller metaphor accurately by telling how the rollers can be rolled, but without mentioning other output modalities. Only domain objects, actions, and visual aspects were judged to be present. Participant of drawing D also explained the roller metaphor accurately, but extended the explanation with references to the physical devices and how roll movements of the ball were aligned with the rotation of the rollers. Only domain objects, actions, physical devices, and visual aspects were judged to be present.

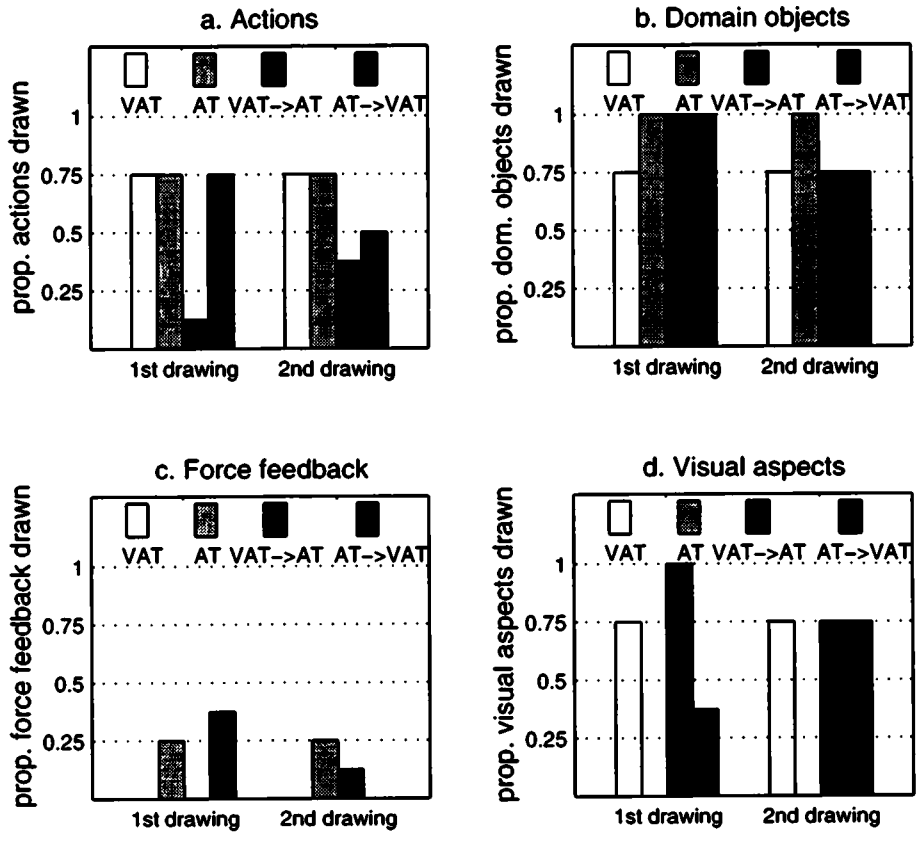

Figure 36. Proportions of aspects in the drawings. over the four conditions. Panel (a) shows the proportion action--related aspects in both drawings. Panel (b) shows the proportion domain object-related aspects in both drawings. Panel (c) shows the proportion force feedback-related aspects in both drawings. Panel (d) shows the proportion visual aspects in both drawings. Recall the experimental conditions have twice as much participants than the control conditions.

10.6.5 Structure diagrams

The choices for the structure diagrams in the four conditions are shown in Figure 37. Recall that all structure diagrams were assigned to a category, denoted by an integer value from 0 through 3 (see also Figure 28). Category 0 contains the correct diagram, whereas category 3 contains the most incorrect diagram.

It appeared that participants in the control conditions were consistent in choosing a structure diagram. This is less true for participants who experienced a change in visual display conditions, i.e., the experimental conditions. However, if only the choices for a correct diagram (diagram category 0) are considered, a test on differences between non-independent proportions did not reveal any possible improvement when participants had experienced a change in visual display conditions ($z = 1.732$).

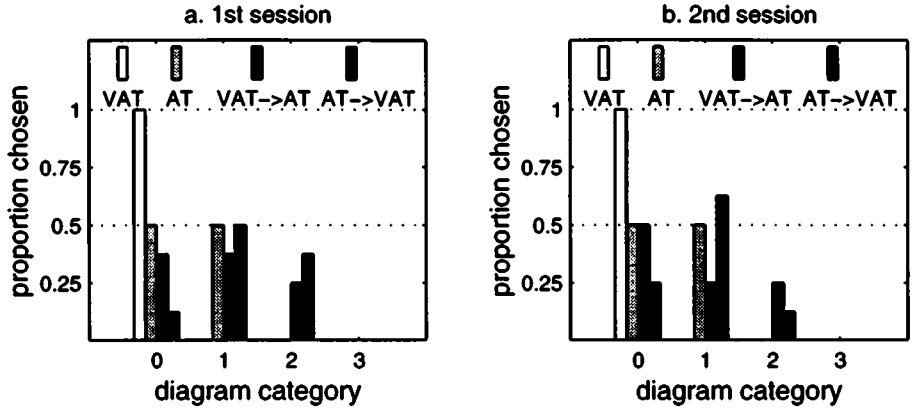

Figure 37. Proportion of chosen structure diagrams over the four conditions. Panel (a) shows the proportion chosen structure diagram at the end of the first experimental session. Panel (b) shows the proportion chosen structure diagrams at the end of the second experimental session. Recall the experimental conditions have twice as much participants than the control conditions.

In the first session, more participants (7 out of 12) who had worked with a visual display choose the correct diagram than those who had worked without a visual display (3 out of 12). A Fisher exact probability test however did not reject the null-hypothesis of equal proportions for both groups ($p = 0.107$). Thus, working with or without a visual display has no effect on choosing the correct structure diagram.

10.6.6 Spontaneous remarks from participants

The participants were invited to express their comments on the interaction style in a spontaneous way. In total, 15 participants took the opportunity to convey some comments by writing down a few words. Two persons only wrote that they were enthusiastic about the interaction style. Seven participants gave no remarks. This section is devoted to a analysis by counting and grouping the written suggestions for improvement.

- *User control.*

  Force feedback. Three participants responded that they had experienced the presence of force feedback as too prominent. As a result, they found the control too demanding for regular use at home.

  Double-presses. Six participants responded that they had experienced trouble with double-pressing the trackball, though participants were given a small tutorial and were notified of the critical action of double-pressing both by instruction and practice (the trackball was not allowed to be moved during the interval inbetween two rapidly adjoining presses).

•*Auditory feedback.* One person found auditory feedback unnecessary when one has the disposal of visual representation. Two persons remarked that several audio events (e.g., speech utterances, non-speech audio) were audible at the same time which made listening to the relevant information harder.

Speech synthesis. Three participants had complaints about the quality of the speech synthesis. One person found the speech unintelligible. Another person would like the synthesis to sound more natural. Yet another person found the volume level of the speech too low with respect to the music.

Non-speech audio. Two persons would like a more prominent sound to notify the event of adding or removing a piece of music to or from the music programme.

•*Synchronization.* Two persons remarked that the coupling between the actions, they executed on the trackball, and the feedback they received from the various devices, was not as direct as it could or should be. They experienced latencies between action and effect and even among feedback events which felt rather "crumbly" to them. Moreover, they felt obliged to wait for events. As a result, they were unable to "act ahead" in order to perform the task as quickly as possible as they were instructed to do.

•*Visual representation.* Five participants found the direction in which the visually represented rollers revolved counterintuitive. Rolling the trackball forth or back resulted into a rotation of the roller in an upward or downward direction respectively. One person was not happy with the graphical representation of the interaction style because of the lack of overview; one cannot disclose the content of the whole music collection by a simple look.

•*Miscellaneous.* Two persons asked for additional functionalities such as scanning the musical content, e.g., fast forward and rewind, additional information on the content of the music pieces, or provisions to change the order of music pieces in a music programme.

10.7 Discussion

The first hypothesis stated that the interaction with a visual display was more efficient than the interaction without a visual display. However, no significant effects in both compilation time and number of actions were observed between visual and nonvisual interaction, except for the first music programming task. Participants who performed the first task without a visual display required more time to complete a music programming task (approximately one additional minute). They did not need to execute substantially more actions. In general, participants learned to perform the tasks more efficiently in both conditions; they needed less time and fewer actions for each successive task.

The second hypothesis stated that users who had worked without a visual display scored higher on procedural knowledge than users who had worked with a visual display. Indeed, users who had worked without a visual display at the first experimental session were better in answering questions about interactive procedures. Also, participants who were transferred to a condition without a visual display at the second exper-

imental session tend to improve their answers to these questions. In general, participants improved in answering questions on interactive procedures for the second time.

The third hypothesis stated that users who had performed two preceding tasks with a visual display performed more efficiently without a visual display than users who started to perform the tasks without a visual display. No significant effects caused by a change in visual display conditions were found for compilation time. However, participants, who had performed two preceding visual interactions, executed significantly fewer actions and spent considerably, but not significantly, less amount of time to perform the tasks without a visual display. It was shown that this effect can entirely be ascribed to practice. It appeared that users who were unfamiliar with the visual display performed less efficient only because they had consequently performed fewer music programming tasks.

The fourth hypothesis stated that users who had worked with a visual display made a personal re-production of the interaction style that contains more visual aspects, whereas users who had worked without a visual display made a re-production which contains more action-and-effect related aspects. It appeared that participants who had worked without a visual display indeed drew fewer visual aspects, and indeed drew more action-related aspects. In addition, a change in visual display condition changed the drawing of visual aspects. It appeared that once participants were visually exposed to the roller metaphor of the interaction style, they were favoured by this metaphor when explaining the interaction style to others. Further, it appeared that the drawings between both conditions of participants were qualitatively different. Participants who were unfamiliar with the interaction concept devised their own metaphor.

An assumption was that nonvisual interaction can not happen without a internal representation. This internal representation should, at least, contain a spatial organization of the objects of interest specified by their interrelations, i.e., a cognitive map. This imaginary to navigate has to be developed mainly during the course of interaction. It is typical that nonvisual interaction requires significant more time and more, but not significantly more, actions to complete a *first* music programming task than visual interaction, probably due to additional effort to develop an internal representation. These effects tend to disappear in successive tasks. In this experiment, it appeared that nonvisual navigation takes more time per action than visual interaction in the first task (nonvisual interaction: 3.2 seconds per action, visual interaction: 1.9 seconds per action). This time difference might be partly ascribed to extra time for cognitive processing to examine the nonvisual percepts resulting from an action, and incorporating them into some internal representation. If we look more closely at the kind of actions that are executed with the trackball at the first task (see Table 5), we observe more lateral roll movements in nonvisual interaction than in visual interaction. These lateral movements are actions to hop from one roller to the other roller. These actions represent navigational act and are apt to explore the spatial relations between the objects of interest. The execution of more lateral movements with the trackball in nonvisual interaction may indicate the required effort in explorative behaviour to explicitly develop a cognitive map of the interaction style. These observations stress the importance of building up an internal representation of the environment in which one interacts to act nonvisually and purposefully in that environment.

Table 5. Mean number of actions categorized in single clicks, double clicks, lateral roll movements, and forward/backward roll movements with the trackball for visual and nonvisual interaction at the first task.

Interaction	Single clicks	Double clicks	Lateral moves	Forward/ backward moves
Visual	5.58	10.0	7.92	27.0
Nonvisual	8.64	10.45	16.82	25.45

As users were assumed to be forced to explore the spatial organization of the interaction style in nonvisual interaction in order to explicitly reveal and memorize actions pertaining to navigation and manipulation, it was expected that they ultimately develop a substantial body of procedural knowledge. Indeed, the experiment demonstrated that participants who performed the task without a visual display of information were better in answering questions on interactive procedures. Making the task more difficult by removing visual display of information induces indeed a higher cognitive load, but excels the learning of procedures.

Another assumption was users internalize and apply the visual conceptual model after visual exposure. Subsequently, it was expected that users use this conceptual model for explaining the interaction style to others. The drawings for explanation of participant who were familiar to the visual roller metaphor contained visual aspect pertaining to this metaphor. In contrast, drawings of participants who were not familiar with the visual conceptual model were qualitatively different; their drawings contain private metaphors and analogies; action-related aspects were more prominent in the drawings. The experiment indeed showed that participant were favoured by using the visual representation of the interaction style in their explanation, once they were visually exposed to the visual representation. However, adopting the conceptual model as a form of foreknowledge did not improve their task efficiency in nonvisual interaction afterwards.

10.8 Conclusion

The purpose of the experimental evaluation was to assess the usability properties of the multimodal interaction style for music programming, in particular, on the presence or absence of a visual display combined with a tactual and auditory interface. The experiment demonstrated that users were able to acquire such a proficiency after three minutes of free exploration and without procedural instructions that they could complete a given music programming task effectively in both visual play conditions. In addition, they learned to perform the tasks more efficiently in both conditions; they needed less time and fewer actions for each successive task.

Though using the interaction style without a visual display involves a considerable cognitive load, users who work without a visual display were ultimately able to perform not significantly less efficiently than users who work with a visual display. Only a little practice will do, and foreknowledge about the visual display is not strictly necessary. One person preferred the displayless interface, because "it contains all attractive fea-

tures that a novel has and a movie picture lacks; one is able to make an own interpretation and devise an own world." All others, who had worked with both versions, preferred a visual display, just for convenience.

Retention tests, for which participants return for re-using the interactive system after a specified amount of time, are often proposed for measuring memorability (Nielsen, 1993). Although participants had no difficulties in re-using the interaction style at the second experimental session, the time for return in this experiment was too short and uncontrolled to make valid conclusions. Memory tests by questionnaire are another means to measure memorability (Nielsen, 1993). The participants in this test were good at completing a questionnaire containing questions about procedures. However, assessing memorability by a questionnaire assumes that users learn to operate an interactive device explicitly, i.e., that they are able to justify their actions and decisions while operating the device, and that they are able to verbalize the acquired knowledge about the device afterwards. Unfortunately, the basic idea that users always learn explicitly is incorrect. For instance, the underlying structure, i.e., the syntax, of the interaction style is deemed to be learned, to a large extent, implicitly and incidentally. A considerable body of research is devoted to the distinction between implicit and explicit learning to control an interactive device and the associated verbalizable knowledge (see, e.g., (Berry and Broadbent, 1984; Berry and Dienes, 1993; Dienes and Fahey, 1998). The general observation in a particular class of control tasks is that practice improves task performance but has no effect on the ability to answer related questions, whereas verbal instruction improves ability to answer questions but has no effect on task performance. Nielsen (1993) argues that the modern graphical user interfaces do not require users to actively remember relevant aspects of the interaction, since the interface reminds users of such aspects when necessary. In fact, Mayes, Draper, McGregor, and Oatley (1988) showed that users were unable to recall the contents of graphical menus after conducting tasks, though they were able to use the same menus when doing the tasks. Summarizing, another experiment controlled to assess memorability has to be conducted to make any valid conclusions on memorability.

Some usability issues concerning user control of the force feedback trackball must be emphasized here. A familiarization phase with the trackball was required to avoid mixing up the experiment; novices are unfamiliar with the affordances of a force feedback trackball. Despite a short introduction on manipulating the force feedback trackball, double-clicking the trackball remained a serious usability problem. This problem is considered a mere software control deficiency to be repaired easily. However, the challenge to minimize the number of control elements in an interaction style by using a force feedback trackball should be considered cautiously. Overloading the trackball with interactive features may have a detrimental effect on the usability of an interaction style.

The visual conceptual model of the interaction style consists of a real-world analogy, i.e., a roller metaphor of a slot machine. Familiar analogies may form a starting point to comprehend an interaction style. As the actions simply happen in the analogy, the effects of actions are directly perceivable and directly interpretable. This 'direct engagement' is assumed to eliminate the *referential distance* in user-system interaction (Hutchins, 1989). Referential distance refers to the gap that exists between the way actions have to be expressed and the meaning of these actions. This distance is assumed a cognitive principle underlying the apparent usability of graphical user interfaces (Hutchins, 1989). In order to establish a *multimodal* interaction style from a real-world analogy and

that should support 'direct engagement', the implementation should not violate human's parsimonious principle of perceiving and subsequently interpreting the world. This principle states that humans interpret percepts, that occur close in time to each other, as being caused by the same effect. Consequently, the software architecture underlying a multimodal interaction style should allow timely synchronization between an action and its multi-sensory effects. However, personal observation and remarks from two participants revealed that there is still some room for improvement to synchronize action and effect.

In this study, it is showed that the tactual and auditory modality can compensate for the lack of a visual display in a multimodal interaction style. The results demonstrate the added value of tactual and auditory feedback in interaction styles used in contexts-of-use in which visual display of information is impoverished or even lacking. Consequently, applications in which multimodal interaction styles can be particularly designed for are portable devices, remote controls, and car equipment. The applications do not have to be strictly focused on music selection and programming, but may also address simple control tasks such channel switching, volume setting, or the selection or programming of other media and content. As nonvisual interaction still involves a considerable amount of cognitive load, results of this study should be carefully considered in multiple tasks paradigm in which attention has to be devoted to an dangerous main task, e.g., car radio and navigation. Summarizing, the results of this study justify a continuing research effort in applying tactual feedback, e.g., by force feedback devices, and auditory feedback, e.g., by speech synthesis and auditory icons, in interaction concepts for user interfaces.

References

Akamutsu, M. (1991). The influence of combined visual and tactile information on finger and eye movements during shape tracing, *Ergonomics, 35,* 647-660.

Akamatsu, A., and Sato. S. (1994). A multi-modal mouse with tactile and force feedback, *International Journal of Human-Computer Studies, 40,* 443-453.

Akamatsu, A., and MacKenzie, S.I. (1996). Movement characteristics using a mouse with tactile and force feedback, *International Journal of Human-Computer Studies, 44,* 483-493.

Akamatsu, M., MacKenzie, S.I., and Hasbrouc, T. (1995). A comparison of tactile, auditory, and visual feedback in a pointing task using a mouse-type device. *Ergonomics, 38,* 816-827.

Anderson, J.R. (1983). The architecture of cognition, Cambridge, MA: Harvard University Press.

Bargen, B., Donnelly, P. (1998). Inside DirectX. Microsoft Press.

Berry, D.C., and Broadbent, (1984). On the relationship between task performance and associated verbalizable knowledge. *The Quarterly Journal of Experimental Psychology, 36A,* 209-231.

Berry, D.C., and Dienes, Z. (1993). Implicit learning: Theoretical and Empirical Issues. Lawrence Erlbaum Associates.

Buxton, W., Gaver, W.W., and Bly S. (1992). The use of non-speech audio at the Interface. Cambridge University Press.

Carroll, J.H., & Olson, J.R. (1988). Mental models in Human-Computer Interaction. *Handbook of Human-Computer Interaction.* M. Helander (ed.), Elsevier Science Publishers B.V. p 45-65.

Cook, R., Morton, B. (1994). The Penguin Guide to Jazz on CD, LP & Cassette, Penguin Books.

Dienes, Z., and, Fahey, R. (1998). The role of implicit memory in controlling a dynamic system. *The Quarterly Journal of Experimental Psychology, 51A (3),* 593-614.

Douglas, S.A., and Mithal, A.K. (1997). The Ergonomics of Computer Pointing Devices. Springer-Verlag.

Eggen, J.H., (1993). Sound at the User Interface. IPO-report 924.

Eggen, J.H., Haakma, R., Westerink, J.H.D.M. (1996). Layered Protocols: hands-on experience. *Int. J. Human-Computer Studies,* 44, 45-72.

Eggen, J.H., and Pauws S.C. (1997). A user-oriented multimedia presentation system for multiple presentation items that each behave as an agent, *Philips patent,* PHN 15.703.

Engel, F.L., Goossens, P.H., and Haakma, R. (1994). Improved efficiency through I- and E-feedback: a trackball with contextual force feedback, *International Journal of Human-Computer Studies, 41,* 949-974.

Engel, F.L., Haakma, R., and van Itegem, J. (1990). Trackball with Force Feedback. Philips patent PH-N 13522, 1990.

Erlewine, M., Woodstra, C., Bogdanov, V. (1994). All Music Guide: the best CDs, LPs & tapes, Miller Freeman Books.

Evans, G.W., and Pedzek, K. (1980). Cognitive mapping: Knowledge of real-world distance and location information. *Journal of Experimental Psychology: Human Learning and Memory, 6,* 13-24.

Freudenthal, D. (1998). Learning to Use Interactive Devices; Age Differences in the Reasoning Process, *Doctoral Thesis,* Eindhoven University Of Technology.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Gaver, W.W. (1997). Auditory Interfaces. *Handbook of Human-Computer Interaction.* 2nd edition. M. Helander, T.K. Landauer, P. Prabhu (eds.), Elsevier Science, pp. 1003-1041.

Gentner, D. and Stevens, A.L. (1983). Mental Models. Hillsdale, NJ: Lawrence Erlbaum Associates.

Gourdol, A., Nigay, L., Salber, D, and Coutaz, J. (1992). Two case studies of software architectures for multimodal interactive systems: VoicePaint and Voice-enabled Graphical NoteBook, *Engineering for Human-Computer Interaction,* Eds. J. Larson and C. Unger, Elsevier Science Publishers, 271- 283.

Halasz, F.G. & Moran, T.P. (1983). Mental models and Problem Solving in Using a Calculator. *CHI'83 Preceedings.* p. 212-216.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming, 8,* 231-274.

Harel, D. (1992). Biting the silver bullet. *Computer (Jan. 1992),* 8-20.

Hauptmann, A.G., and McAvinney, P. (1993). Gestures with speech for graphic manipulation, *Internation-*

al Journal on Man-Machine Studies, 38, 231-249.
Hermes, D. (1998). Auditory Material Perception. Submitted to ICAD'98, Glasgow.
Hutchins, E. (1989). Metaphors for Interface Design. *The Structure of Multimodal Dialogue.* eds. M.M. Taylor, F. Néel, and D.G. Bouwhuis. Elsevier Science Publishers.
Keyson, D.V., and Tang, H. (1995). TacTool: A tactile rapid prototyping tool for Visual Interfaces. Symbiosis of Human and Artifact, *Proceedings of the 6th International Conference on Human-Computer Interaction,* Tokyo, pp. 67-74. Amsterdam-Elsevier.
Keyson, D.V. (1996). Touch in User Interface Navigation, *Doctoral Thesis,* Eindhoven University of Technology.
Keyson, D.V., and Stuivenberg, L. van (1997). TacTool v2.0: An object-based multimodal interface design platform, *Proceedings of HCI International,* San Francisco, CA, USA, August 24-27.
Kientzle, T. (1998). A Programmer's Guide to Sound. Addison-Wesley.
Kieras, D.E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science,* 8, 255-273.
Klatzky, R.L., Loomis, J.M., and Golledge, R.G. (1997). Encoding Spatial Representations through Nonvisually Guided Locomotion: Test of Human Path Integration. *The Psychology of Learning and Motivation,* VOL. 37, Eds. Douglas L. Medin, Academic Press, p. 41-84.
Kotovsky, K. and Simon, H.A. (1990). What makes some problems really hard? Explorations in the problem space of difficulty, *Cognitive Psychology, 22,* 143-183.
Mayes, J.T., Draper, S.W., McGregor, A.M., and Oatley, K. (1988). Information flow in a user interface: The effect of experience and context on the recall on the recall of MacWrite screens. In Jones, D.M., and Winder, R. (Eds.), *People and Computers IV,* Cambridge University Press, Cambridge, U.K., 275-289.
McNemar, Q. (1962). Psychological Statistics. Third edition. John Wiley and Sons.
Nelson, R.J., McCandlish, C.A., and Douglas, V.D. (1990). Reaction times for hand movements made in response to visual versus vibratory cues, *Somatosensory and Motor Research, 7,* 337-352.
Newell, A. and Simon, H.A. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall.
Nielsen, J. (1993). Usability Engineering. Academic Press.
Norman, D.A. (1983). Some observations on mental models. In D. Gentner & A.L. Stevens Eds. *Mental Models.* pp. 15-34, Hillsdale, NJ: Lawrence Erlbaum Associates.
Pan, D. (1995). A Tutorial on MPEG/Audio Compression. *IEEE MultiMedia, Vol. 2, No. 2,* 60-74.
Pauws, S.C., and Eggen, J.H. (1996). New functionality for accessing digital media: Personalised automatic track selection, In: A. Blandford and H. Thimbleby (Eds.) *HCI-96, Industry Day & Adjunct Proceedings,* Middlesex University, UK.
Pauws, S.C., Ober, D., Eggen, J.H., and Bouwhuis, D.G. (1996). A comparative evaluation of strategies for compiling music programmes. *IPO Annual Progress Report,* 31, 50-58.
Pauws, S.C., Eggen, J.H., and Bouwhuis, D.G. (1997). Explorative strategies while compiling music. *IPO Annual Progress Report,* 32, 79-88.
Payne, S.J., Squibb, H.R., and Howes, A. (1990). The nature of device models: The yoked state space hypothesis and some experiment with text editors. *Human-Computer Interaction,* 5, 415-444.
Presson, C.C. (1987). The development of spatial cognition: Secondary uses of spatial information. In N. Eisenberg (Ed.), *Contemporary topics in development psychology* (pp. 7-112). New York: Wiley.
Rasmussen, J. (1986). Information processing and human-machine interaction: An approach to cognitive engineering. Amsterdam: North-Holland.
Robbe, S., Carbonell, N., and Valot, C. (1997). Towards usable multimodal command languages: Definition and ergonomic assessment of constraints on users' spontaneous speech and gestures. *Eurospeech '97,* 1655-1658.
Rogerson, D. (1997). Inside COM. Microsoft's Component Object Model. Microsoft Press.
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). Object-Oriented Modeling and Design. Prentice-Hall International.
Sholl, M.J. (1996). From visual information to cognitive maps. In J. Portugali (Ed.), *The construction of cognitive maps* (pp. 157-186). The Hague: Kluwer.
Siegel, S. (1956). Nonparametric Statistics. McGraw-Hill.

Staggers, N. and Norcio, A.F. (1993). Mental models: concepts for human-computer interaction research. *International Juornal of Man-Machine Studies 38*, p. 587-605.

Thorndyke, P.W., and Hayes-Roth, B. (1982). Difference in spatial knowledge acquired from maps and navigation. *Cognitive Psychology, 14,* 560-589.

Van der Veer, G.C. (1994). Mental models of computer systems: visual languages in the mind. In Tauber at all (ed.): *Cognitive aspects of visual languages and visual interfaces*. Elsevier Science, Amsterdam.

Woodworth, R.S., & Schlosberg H. (1965). Experimental Psychology (3rd ed.). Methuen & Co. Ltd.

Young, R.M. (1981). The machine inside the machine: Users' mental models of pocket calculators. *International Journal of Man-Machine Studies*, 15, 51-85.

Appendix I Instruction text

In order to provide declarative knowledge, participants were informed about the concepts in the music programming domain by means of the following text (translated from Dutch).

> The aim of the experiment is to compile a music programme by means of an interactive system. A music programme contains a list of 10 pieces of music, that is compiled on the basis of your personal preferences. Therefore, the compilation of a music programme demands the making of music choices. The interactive system enables you to add pieces of music to your music programme or, if desired, to remove pieces of music from your music programme in order to conclude with a preferred music programme.
>
> The pieces of music can be chosen from a music collection, consisting of 480 pieces of music, in total, recorded by a variety of jazz musicians. The music collection is divided into 12 popular jazz styles; each jazz style comprises 40 pieces of music and each piece belongs to only one jazz style. An exception to this rule is determined by the definition of a jazz style called 'all styles' in which all other jazz styles, and, thus, all 480 music pieces, are collected. The interactive system enables you to select a particular jazz style, through which you can choose between the music pieces that belong to the selected jazz style.
>
> Beside the music collection, you have the disposal of so-called music recommendations. A recommendation is a piece of music that also resides in the music collection. Each piece of music in the music collection has attached to it 4 recommendations, that are provided by the interactive system on the basis of some commonality. It is unimportant to know on what grounds this commonality is based, but you can trust that a piece of music and its 4 recommendations belong to one and the same jazz style. The interactive system enables you to add recommendations to your music programme.
>
> At all times, you are allowed to get informed by the interactive system to what jazz style a particular piece of music belongs, or what jazz style is selected. Music is played back automatically.

At the outset of each music programming task, instruction about the task was provided by means of a text. Important criteria were stressed by emphasizing, underlining, and bold-facing words in the text. Four different task instructions were used that differ only in the jazz styles the participants were restricted to select from. The following text was provided (translated from Dutch).

> You are given the opportunity to compile a music programme by means of the interactive system. Your music choices may be selected from the music collection as well as from the music recommendations.
>
> However, three requirements apply, that your personally compiled music programme *has to* satisfy.
>
> 1. The music programme has to contain exactly 10 pieces of music.
>
> 2. All pieces of music in the music programme have to be distinct.
>
> 3. The music programme has to contain pieces of music from 2 different jazz styles:
>
>   a. 5 pieces of music from blues (mbase, bebop, neobop)
>   b. 5 pieces of music from swing (dance, hardbop, fusion)
>
> You are instructed to compile your music programme **as quickly as possible**. It is **not** important that the pieces of music in the music programme are preferred. Also, the order of pieces within the music programme is **not** important.

Appendix II Transcription device

The level of procedural knowledge is assessed by means of a questionnaire containing question about small interactive procedures. In order to transcribe the answers, the test supervisor used a device for constructing augmented regular expressions over *actions terms*. All eight actions were represented by a symbol which were combined with the following operators: ( ) - parentheses, • - sequence, + - selection, * - repetition (in decreasing order of precedence). The symbols are shown in Table 6. The semantics of the augmented regular expressions are summarized in Table 7, where X and Y are arbitrary action expressions.

Table 6. The actions represented by symbols as defined by the transcription device.

symbol	action
←	roll to the left
→	roll to the right
↑	roll forward
↓	roll backward
⇑	roll forward through
⇓	roll backward through
●	click
●●	double-click
?	any unfamiliar action

Table 7. The semantics of the operators as defined by the transcription device.

expression	operator	meaning
(X)	parentheses	X is grouped and overrules order of precedence
X • Y	sequence	X is followed by Y; Y follows X
X + Y	selection	X or Y can be selected
X*	repetition	Zero or more times of X

Procedural knowledge is knowledge about what actions are required for a successful control of the interaction style. Each item of the questionnaire asks what physical action sequence of *minimal* length is required for a given effect. One half of the items addressed procedural knowledge on single actions (one-step interaction); the other half addressed procedural knowledge on action sequences of length 2 or 3 (multiple-step interactions). A one-step interaction comprises the execution of a basic action (e.g., adding a track to the programme), but also the use of compound actions (e.g., a fast scroll mechanism). A compound action is a single action which stands for a sequence of basic actions. Since a

multiple-step interaction involves the formulation of a longer action sequence to achieve a purpose, forward and backward chaining reasoning mechanisms to contrive such sequence are required. However, the one-step interactions items are not considered less difficult to answer; it is assumed that the compound actions are not easily discovered and directly comprehended.

An example of a one-step interaction item (a compound action) reads as follows (translated from Dutch):

> You are currently listening to a music piece in the collection. What action(s) do you need to execute to go as quickly as possible to another piece of music within the collection by a step size of 3?

Although many syntactic variants are possible, a correct answer might sound like:

> A fast hand-stroke executed on the ball in a forward or backward direction covering a somewhat longer distance.

The test supervisor had to transcribe this answer as follows

> ⇑ + ⇓

An example of a multiple-step interaction item reads as follows (translated from Dutch):

> You are currently listening to a recommended piece of music. You have just added two distinct recommended pieces of music to your music programme. The music programme contains two pieces of music. What action(s) do you need to execute to remove as quickly as possible the firstly added piece of music from your music programme?

A correct answer to this question consists of no more than three actions that might sound as follows.

> Roll the ball to the right, until you 'fall into' the music programme. Then, roll the ball back or forth, only once, to get at the firstly added piece of music. Conclude with a double-click on the ball to remove the piece.

The transcription of this verbal protocol is as follows

> → • (↑ + ↓) • ●●

In order to derive a quantitative measure of the level of procedural knowledge, all transcribed action sequences were counted that met a pre-defined *correctness* criterion. Action sequences were considered correct if they were effectivity, i.e., whether the action sequence led to the intended result, and if they were efficient, i.e., if the action sequence was of minimal length.

Appendix III Pre-analysis of task performance measures

The distributions of the number of actions and compilation time were positively skewed; some participants required an extreme high value in effort and time to complete a task. The possible causes of these extreme cases had to be revealed. A first suggestion is that some participants had not obeyed the task description as intended. Although the task objective was to compile a music programme as quickly as possible without taking care of personal music preference, it is likely that some participants were, in spite of it, tempted to select preferred music, and consequently explored the music collection in search for good music.

The number of actions is a summation of all kind of actions on the trackball device: forward and backward roll movements within the four rollers, lateral roll movements from one roller to another, single clicks, and double clicks. More specifically,

$$\text{number of actions} = \#clk + \#dclk + \#nvg + \#prg + \#sts + \#msc$$

where #clk denotes the number of single clicks, #dclk denotes the number of double clicks, #nvg denotes the number of lateral roll actions for navigation, #prg denotes the number of roll actions within the programme roller, #sts denotes the number of roll actions within the styles roller, and #msc denote the number of roll actions within both the music collection and recommendation rollers.

Table 8. The number of distinct actions per effort category.

effort category	#task	#trks	time	#clk	#dclk	#nvg	#prg	#sts	#msc	mean effort	time per action
ideal	.	10	.	0	10	3	0	2	8	23	.
23-30	7	10.6	59.9	1.86	10.0	3.43	0	5.29	8.14	28.71	2.09
31-40	32	13.1	82.6	4.84	10.06	5.59	0.06	6.94	8.84	36.34	2.27
41-50	22	16.2	112.1	9.45	10.27	7.45	0.59	7.36	10.45	45.59	2.46
51-60	10	23.4	155.6	11.2	10.6	7.8	1.6	9.3	16.1	56.6	2.75
61-70	9	28.1	162.7	12.11	11.33	14.33	2.78	11.22	14.0	65.78	2.47
71-80	4	41.3	185.3	8.75	10	14.25	2	13	27.75	75.75	2.45
81-90	4	35.3	260.3	12	10.75	19	2.75	19.5	20.5	84.5	3.08
91-100	2	47	285.5	12	12	24	3.5	15.5	24.5	91.5	3.12
>100	6	63.8	343.8	25.67	11	23.83	0.67	20.83	46.83	128.83	2.67
	96	22.1	135.1	8.94	10.41	9.35	0.9	9.38	14.37	53.35	2.53

An inventory of what kind of actions constitutes the total number of actions and how these constituent actions behave at an increasing number of actions is made and shown in Table 8. For that, task executions were assigned to an *effort category* indicating a typical range of the number of actions executed. Each category was assigned a certain

number of task executions (#task). Executing a task with the minimum of 23 actions was considered ideal. While executing a task, participants came across, i.e., listened to, a certain number of pieces of music (#trks). Also, they spent a certain amount of time, i.e., compilation time, to complete the task (time measured in seconds). Further, the mean number of actions within each effort category is decomposed into its constituents: mean number of clicks (#clk), mean number of double clicks (#dclk), mean number of lateral roll movements (#nvg), mean number of roll movements within a programme (#prg), mean number of roll movements within the styles (#sts), and mean number of roll movements within the music collection and recommendations (#msc). The table concludes with the mean number of actions executed and mean time per action spent (compilation time/mean number of actions executed).

At least, three observations can be made from Table 8. First, the compilation time is proportional to the number of actions executed by a factor of approximately 2.53. The mean time per action remains almost constant over the effort categories (see the column at the right-hand side of Table 8). Hence, irrespective of how many actions are executed, it seems that a fixed amount of time is spent for each single action.

Second, as the number of actions increases, so will the number of music pieces listened to. In particular, #msc, and to a lesser extent, #sts and #nvg are suspected to be caused by this music exploration. These components increase proportionally with increasing number of actions executed, whereas other components remain constant. As the number of actions executed is coupled with the number of music pieces listened to, it is likely that participants who performed relatively many actions actually explored the music collection in search for preferred music. If that is the case, they did not however allocate extra time to music listening and judgement, because the mean time per action remains constant at increasing number of actions executed. It seems that they only took a brief notice of each piece of music, but listened to many pieces successively. Whether participants were actually exploring the music collection or not can not be directly. stated from the raw data, because there were no explicit actions that refer to music listening; music was always audible in the experiment. On the other hand, instead of suspecting participants of music exploration, participants could also experience troubles in performing the tasks. Also these participants are forced to execute a large number actions, and consequently come across a lot of music, rather unintentionally.

Third, the number of single clicks (#clk) is rather erratic in the different *effort categories*. A large number of single clicks can be partly ascribed to faulty executions of double clicks. A faulty execution of a double-click resulted into two unintended single clicks. In fact, six participants explicitly reported that they experienced difficulties in double-clicking the trackball.

Music exploration and unintended single clicks are undesired features in the data. Whereas attenuating the music exploration effect is troublesome, the unintended single clicks can be easily removed from the data by holding only the first single click in a sequence of single clicks, i.e, by replacing each sequence of multiple single clicks by a single execution of a single click. The removed single clicks are thus interpreted as being unintentionally. The time intervals associated with these unintended single clicks are also removed from the total compilation time.

Additional time and actions that are devoted to music exploration can not be traced back from the raw data. However, one particular participant's behaviour was highly suspected to be dedicated to music exploration. The participant, assigned to the AT->VAT experimental condition, spent almost nine minutes (526 seconds) and almost exactly nine minutes (542 seconds) to complete respectively the first and second task, whereas the mean compilation time for all tasks in the condition was 2 minutes and 42 seconds (162 seconds). In addition, the participants performed 155, 175, and 137 actions for respectively the first, second, and third task, whereas the mean number of actions executed was about 59 actions. Because the above mentioned cases were considered outliers (they fell almost 3 times above the interquartile range) and the participant freely admitted to search for preferred music while performing the first three tasks, the data associated with this participant were excluded from the task performance analyses. For the other analyses, the data of this participant were considered still valuable and were left unchanged.