A task analysis of music composition and its application to the development of Modalyser

RICHARD POLFREMAN

Music Department, University of Hertfordshire, College Lane, Hatfield, Herts AL10 9AB, UK
E-mail: R.P.Polfreman@herts.ac.uk

This paper presents an overview of a generic task model of music composition, developed as part of a research project investigating methods of improving user-interface designs for music software (in particular focusing on sound synthesis tools). The task model has been produced by applying recently developed task analysis techniques to the complex and creative task of music composition. The model itself describes the purely practical aspects of music composition, avoiding any attempt to include the aesthetic motivations and concerns of composers. We go on to illustrate the application of the task model to software design by describing various parts of Modalyser, a graphical user-interface program designed by the author for creating musical sounds with IRCAM's Modalys physical modelling synthesis software. The task model is not yet complete at all levels and requires further refinement, but is deemed to be sufficiently comprehensive to merit presentation here. Although developed for assisting in software design, the task model may be of wider interest to those concerned with the education of music composition and research into music composition generally. This paper has been developed from a short presentation given at the First Sonic Arts Network Conference in January 1998.

1. INTRODUCTION

The work described in this paper was born out of a desire to develop improved user-interface designs for music software. In particular, there appeared to be problems with software-based sound synthesis systems which often present users with unfamiliar (to musicians) tasks and notations, requiring the user to be more of a computer programmer and signal processing expert, rather than a composer. The philosophy behind the project was that if we could try to understand better the processes by which composers in general produce musical works, we could then design software that would better match the needs, knowledge and expertise of composers who are not necessarily expert in either computer programming or sound synthesis techniques. The research involved taking user-centred design techniques from the field of human computer interaction (HCI) and applying these to the challenging problem domain of music composition. Specifically, a method of task analysis (TA) known as KAT/TKS (Johnson 1992) was used to produce a generic description of music composition tasks that could then be used to assist the software design process. The generic task model (GTM) developed aided our understanding of the nature of music composition tasks, the environment within which tasks are typically carried out, and how these tasks are organised collectively. A summary of early results can be found in Polfreman and Sapsford-Francis (1995), while a complete description of this research can be found in Polfreman (1997b).

Modalyser, a graphical environment for sound synthesis with IRCAM's Modalys, has been developed using user-interface ideas emerging from the GTM, particularly in terms of its structure and approach to describing musical elements. It is currently a working prototype that has been made freely available to users of Modalys (Morrison and Adrien 1993) via the IRCAM Software Forum. Modalyser is not yet complete in terms of providing all the functionality of Modalys and in terms of providing all the support functions necessary for efficient user interaction. However, the program does serve as the basis for developing a more complete user-interface solution and as a way of gaining user feedback on the general user-interface design.

2. TASK ANALYSIS

Task analysis methods are generally aimed at producing structured models of how people carry out particular tasks. In this research, a recently developed technique, known as KAT/TKS, was applied for the first time to the problem domain of music composition. Little TA research regarding music composition has been carried out before, the principal work being that of Otto Laske in the field of cognitive musicology. Laske states that, '. . . the kind of musical knowledge that, if implemented, would improve computer music tools is often not public or even shared among experts, but personal, idiosyncratic knowledge . . . the elicitation of personal knowledge, and of action knowledge, still awaits a methodology, and easy to use, interactive support tools'. (Laske 1992) While there are parallels that can be drawn between task models in the form of Johnson's task

Organised Sound **4**(1): 31–43 © 1999 Cambridge University Press. Printed in the United Kingdom.

knowledge structures (described below) and Laske's model of musical activity (Laske 1992), the approach to gathering task data, the level of composition task knowledge sought and the resulting models themselves are very different in the two cases. In particular, much of Laske's research has involved detailed studies where subjects are set specific composition tasks and use specifically designed task environments on computer, whereas our work has concentrated on studying real compositional tasks in their usual or natural surroundings. That is to say, we used composers who were writing pieces whether or not we were involved; they were usually observed/interviewed in the places where they were going to be working and we tried to interfere as little as possible in the composition process. Laske's work is also concerned with developing artificial intelligence systems that embody compositional theories and can therefore compose music. This is not what our analysis work had as a goal, and indeed it would be difficult to use our model in such a way, since it is not aimed at describing composition tasks at such a level. Nevertheless, it may be that our GTM may serve as a framework within which cognitive musicological ideas may be further researched and perhaps better understood.

Knowledge analysis tasks (KAT) is a suggested set of methods for producing a generic task model expressed in terms of task knowledge structures (TKSs) which are organised into three main substructures – goal, procedural and taxonomic structures. The goal structure contains *goal* and *subgoal* elements and the control relations between them (which embody *plans*). Goals and subgoals are states of the environment to be achieved, e.g. 'edit sound', 'play note', etc. The procedural structure contains the *procedures* for achieving goals/subgoals in terms of *actions* acting on *objects*. An object in this model is defined by its set of attributes (data) and actions (methods) that can be applied to it. The taxonomic structure contains object definitions and other useful information relating to the objects (such as typical instances, which procedures use the object, etc.). Figure 1 shows a summary of the structure of TKSs.

A TKS is built up on the basis of information gathered from various sources. In this research, questionnaires, interviews with composers and direct observation of composers at work were used as the main sources of data for the analysis. The task model itself was taken back to composers involved in the study to verify that the model represented the composers' tasks adequately and to make amendments where necessary. Composers of different musical styles and using different technologies were used as subjects so as to produce as generic a task description as possible.

3. THE GTM

3.1. Overview

There is not scope within this paper to present the entire TKS, or even the majority of its elements. However, a summary overview of the model is given, and some points of interest further into the model (relating to definitions occurring in the taxonomic structure) are described. A complete description of the model can be found in Polfreman (1997b). Figure 2 shows a network of the top-level goals of the TKS. It should be stated here that with a task as complex as music composition there clearly cannot be a single definitive task model – the model presented here is how we have analysed the task, others may produce very different task descriptions. In the figure, the shadings indicate to which of three main areas a goal belongs – darkest = 'design framework' goals, palest = 'research' goals and mid-shade = 'produce music' goals.

Design framework involves the setting out of what can be seen as a set of constraints within which the piece will be composed. This framework covers both music-related constraints (e.g. instrumentation, musical structure) and practical ones (e.g. tools to be used in the composition process, the final format of the work). This goal can often be the most important part of the composition process, after which point composers sometimes state that the piece is in fact completed, even if no 'notes' have been written down. In other cases, much of this goal is determined with little conscious effort immediately a work is begun.

Research may be necessary before a new work can be completed. It includes many subgoals focused upon particular topics that are typically of interest to composers. These research goals are not analysed any further and it would be difficult to do so in any generic way. However, awareness of these various research areas could be useful in providing support tools within a computer-based music system. In particular, it may be useful to incorporate well-known products of these research areas into software tools (such as the standard pitch ranges of acoustic instruments in a score typesetting program).

Produce music is the goal of setting down the musical material itself in an external form so that it can be performed or tested in some way. This goal involves the creation of the final deliverable product of the composition process, a product which may exist in one (or more) of many different forms. This goal also includes the production of musical sketches and rough drafts that do not necessarily form part of the final artefact.

In general terms, the goals of *design framework* and *research* are carried out, at least partially, before that of *produce music* which leads to the completion

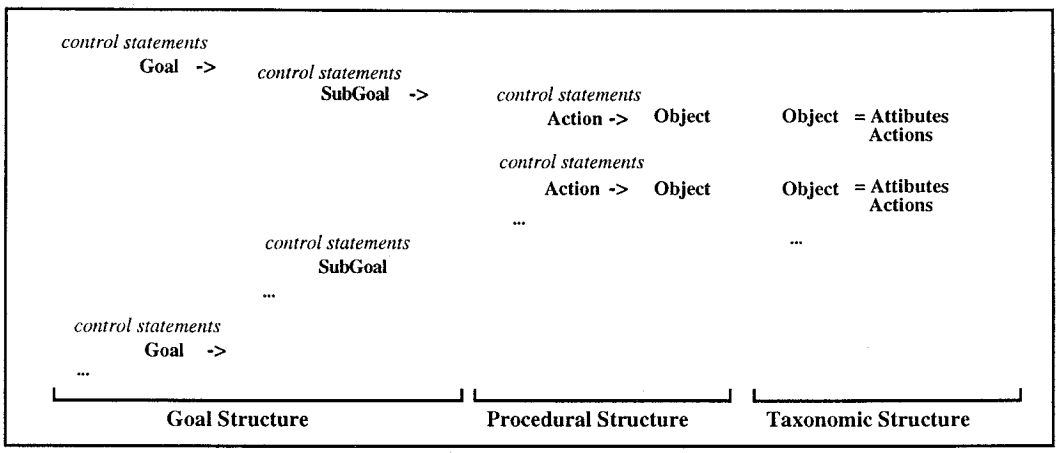


Figure 1. Summary of task knowledge structures.

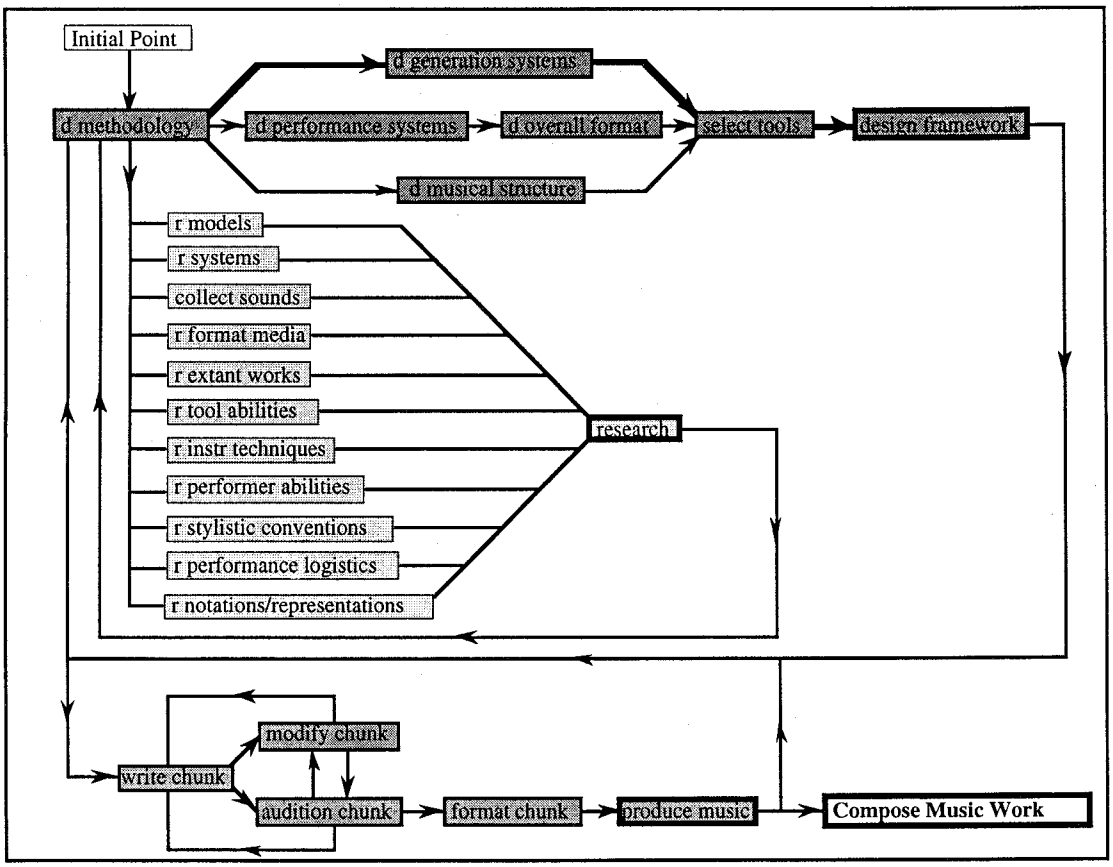


Figure 2. The GTM top-level goals network.

of the piece. However, there is often much interplay between the three regions, and at any time the composer can jump back to earlier points and either complete a task that was not completed, or decide that although completed, the solution produced was wrong and so the task needs to be carried out again. Two important attributes of the goals network need to be emphasised. First, that of *partial completion*. This means that partial completion of a goal allows the partial completion of a subsequent goal, rather than there being a requirement that a goal must be fully complete before moving on to the next goal. Secondly, that a composer may jump to any goal, at any time, provided that the sufficient conditions for commencing work on that goal are satisfied (i.e. that necessary preceding goals have been fully or partially

completed). These two elements, along with the many feedback loops that occur in the model, allow the network to be open enough to cope with the diversity of actual task performance that occurs in music composition.

3.2. Design framework

The starting point is *design methodology*. This involves the formation of a *plan* of action in order to carry forward the composition process. A *plan* is essentially a list of goals to be completed in a certain order (with possible parallelism). A minimal *plan* may consist of only one target goal. *Plans* may (and are likely to) change during the task – the composer returning to this goal. It must be stated that some goals may already have been achieved (consciously or subconsciously) by the time a *plan* for directing the composition process is developed. That is to say, the original conception of, or commission for, a musical work may determine various aspects of a work in advance – such as instrumentation, structure or musical theme. Initially, from *design methodology* the composer moves to either a research goal, or to one of designing *generation systems*, *performance systems* or *musical structure*.

A *generation system* is a scheme for generating musical material at a level above *instrument techniques* (defined a little later in this section). Typical *generation systems* may include: improvising in a particular key; using a mathematical rule to select notes from a defined mode or pitch set; use of trial and error in assembling a series of soundfiles; intuitive pitch selection within harmonic constraints. A *generation system* itself contains two different elements: selection *processes* that generate values, and *constraints* that decide how the values are mapped onto a set of (musical) events. Thus, *processes* include improvisation, random number series, matrices, intuition, extra-musical patterns; *constraints* include keys/scales, modes, metres, duration sets, event ordering rules. The combination of at least one *process* and at least one *constraint* forms a system that produces musical material when activated. A single *generation system* may determine only one particular element of musical material, such as pitch. In this case, other systems must be used to decide durations, dynamics, *instrument techniques*, etc. For example, in the case of one of the observed composers, a graphical pattern matched to a quarter-tone scale was used to determine pitches, while durations and dynamics were left to be derived intuitively. The *constraints* need not be fixed for a particular system. For example, a *constraint* that defines a set of possible pitches might shift upwards by a semitone after every fifth pitch selected. This can be thought of in terms of a *process* (here, a linear ramp function) applied to a *constraint*. It is also possible to apply *processes* to *processes* – for example a simple 'repeat item' *process* could be applied to a graphical pattern. While in a simple *generation system* we might expect one or two *constraints* to be governing a single selection *process*, it is possible to involve many *constraints*, selection *processes*, and *processes* applied to both *constraints* and *processes* in a single *generation system*. The interaction between several simple *processes* and *constraints* can result in complex musical material. While the description of *generation systems* given here may appear very mechanical (except in the case of intuitive systems), subsequent editing of material by a composer can have the purpose of imposing intuitive intervention onto mechanistically derived events. In the case of algorithmic composition, *generation systems* are expressed in mathematical terms by a composer and their development forms a major part of the composer's work. At least one *generation system* must be in place for the music production goal to be achieved, even if this system simply involves writing out from memory musical ideas that have already been formed by subconscious thought processes. This necessity is indicated in figure 2 by the heavier line from *design methodology* through *design generation systems* and *select tools* to *design framework*.

In the task model, a *performance system* is defined as 'a system that given a score produces sound'. (Here the word 'score' has a wide interpretation.) Using such a definition allows the model to be as generic as possible and include a wide range of possible scenarios. A *performance system* includes three major components that are defined as follows:

- *instrument*: a system of sound production that has an associated set of potential sounds;
- *instrument technique*: a method for the production of a subset of an instrument's potential sounds; and
- *performer*: an interpreter that reads a score and activates instrument techniques.

The definitions of these items include further analysis into generic components that make up instruments, techniques and performers, but we will not cover these here. The goal *design performance system* entails defining the group of *instruments* and *instrument techniques* that will be used (the instrumentation) and also the *performers* who will play the piece. *Instruments* in this model may be computer systems, electronic synthesizers and signal processors, in addition to acoustic and electroacoustic instruments. *Instrument techniques*, as stated above, are methods of activating subsets of instrument's potential sounds and as such include bowing, plucking, striking, sound samples, synthesizer patches, etc., while *performers* 'interpret' scores and activate these techniques at appropriate times and with appropriate parameters.

In electroacoustic works, designing instrumentation generally becomes a key focus for a piece and there is a great deal of interaction between this goal (where sounds are designed) and subgoals of *produce music* (where sounds are arranged). In many acoustic pieces this goal is simply a early compositional decision (perhaps dictated by a commission) such as employing a standard string quartet and traditional string instrument techniques. However, much contemporary acoustic work deals with developing new *instrument techniques* – this part of the performance system being a major compositional task. Figure 3 compares some example performance systems.

A composer may, of course, use various performance systems for a particular musical work, some purely for aid during the compositional task rather than for final performance (e.g. a score typesetting program and MIDI sound module may be used in the composition of a work for acoustic instruments). Figure 4 shows a summary definition of 'instrument' taken from the taxonomic substructure of the GTM.

Design musical stucture refers to the creation of a skeletal (temporal) framework within which the musical material produced by *generation systems* will be assembled. This relates to the overall shape of a piece as well as internal relationships between material occurring at different locations (score parts and/or times). The level of detail that a composer consciously works into these structures is highly varied. In questionnaires, some composers indicated a top-down approach to composition, whereby the structure is developed and refined to a detailed state before material is written. Others indicated a bottom-up approach, where the generation of material leads to an emerging structure. The interaction between this goal and that of designing *generation systems* is very important – in fact the distinction between the two areas can become blurred. For example, in some cases *generation systems* may be applied recursively to material, first defining events, then organising these progressively to form structural hierarchies. We must be clear though, that a *musical structure* can exist independently of whether any musical material has been written or not. Here, we define a *musical structure* to be constituted from a hierarchy of *structural components* and *structural relations*. A *component* is simply a subunit of a *musical structure* (which may itself contain various *components* and *relations*), while a *relation* expresses some kind of aesthetic and/or musical link between components. Components and relations may be very simple (a component may be a container for a simple melody; a relation could indicate that one component is a transposed version of another) or can be complex (a component representing an entire movement of a work; a relation that involves many mathematical transformations applied successively). It is possible that a composer may be working with different musical structures for the same piece simultaneously, each structure looking at the work from different musical or aesthetic viewpoints.

Following from *design performance systems* is a related goal – *design overall format*. Overall *format* refers to the nature of the deliverable materials of the composition process – printed scores, tapes (DAT, analogue), signal processor routings, etc. This *format* is usually dependent upon the **types** of the instruments used, if not the actual instruments themselves. A *format* itself covers both the physical media upon which information is stored and also the information representation (and rendering thereof) used. Thus, a typical printed score *format* might consist of A3 paper, using CMN representation and a standard Visually Rendered Notation (VRN) for CMN (Huron 1992). The overall *format* is usually decided early on, but not necessarily precisely. For example, there may be the idea of using multitrack tape in a work, but whether it is analogue or digital and what particular tape speed or digital format is to be used may be decided later.

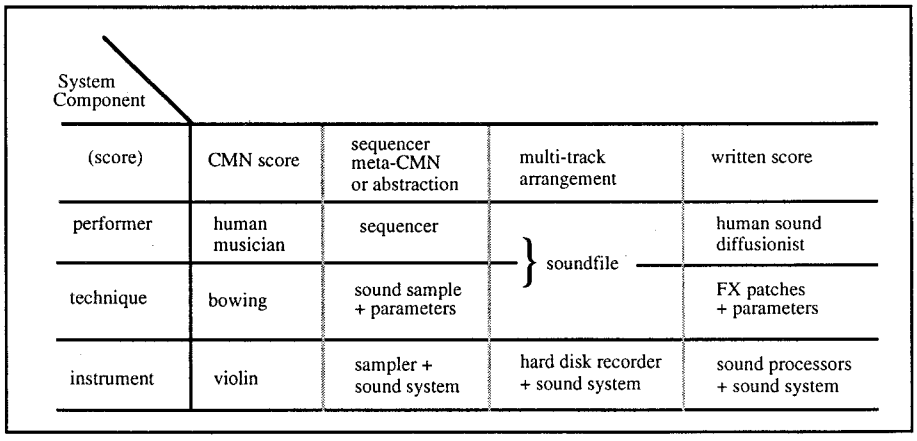


Figure 3. Examples of *performance systems* (types of corresponding scores also shown).

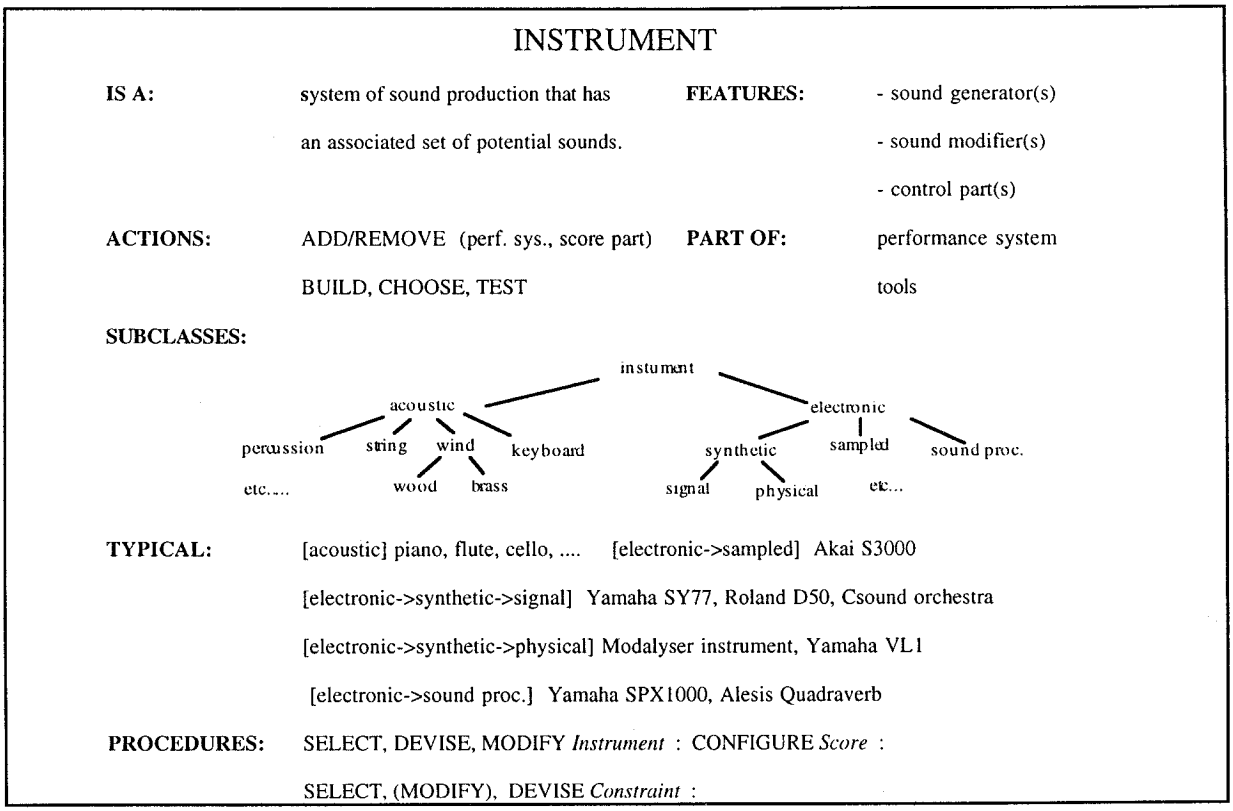


Figure 4. Instrument definition from the taxonomic substructure.

From *design generation systems*, *design musical structure* or *design format*, the composer moves to *select tools*. The *tools* consist of any items to be used in the composition task – musical instruments, writing implements, reference materials, computer software, etc. The idea of partial completion is important here, since tools may be necessary for use in previous goals – e.g. *design generation systems*. In this case, some decision is made about the type of *generation system* to be used, *tools* are then selected for the design process and then the composer returns to *design generation systems* to complete the actual design. The *select tools* goal is placed after these other goals since normally some earlier decisions must be taken, before the *tools* for the task can be chosen.

3.3. Produce music

The primary goal of *design framework* is achieved by completion of the goals described above. From this point the composer can move on to the subgoals of *produce music* or return to *design methodology* or any (partially) completed subgoal covered so far. The first subgoal of *produce music* is *write chunk*. A *chunk* in this model is a defined segment of musical material of arbitrary size, usually (but not always) bounded in some musically meaningful way. A *chunk* may be purely conceptual in its delineation, rather than explicitly set out in the music. A *chunk's* size can range from a single note event, through gesture and phrase up to section, movement or entire work. A *chunk* may be assigned temporarily by a composer to a region of music or may be a significant fixed feature within the structural framework of a piece. A composer's work at any one time is usually concentrated on a particular *chunk*, rather than scattered arbitrarily through a piece of material. There are parallels between the *chunks* that make up a piece and the musical structure of a work, but not necessarily a direct correspondence. That is to say, a chunk that has been written may realise a structural component in a work, but may just be a part of a component or can be spread across several components. A chunk itself is produced by activating one or more *generation systems* to produce the material.

Once a *chunk* has been written, the composer may proceed to *audition chunk* or *modify chunk*. Auditioning is a checking mechanism to see if a *chunk* is 'correct' (i.e. meets current musical requirements) and if not, to identify the precise faults. This can be done via visual (i.e. score reading), or aural (i.e. sound playback or pure imagination) means, or a combination of the two. Since 'current musical requirements' may change at any time, *chunks* are often auditioned repeatedly at different times during the composition process. Modifying a *chunk* can be achieved in a variety of ways. Two broad categories are: the application of a global transformation to a

chunk – such as reverse, transpose, invert, delete; the location of a sub-*chunk* within a *chunk* and subsequent transformation of this sub-*chunk*. Auditioning can also lead to changes in *performance systems* rather than changes to the *chunk* material itself – particularly in the case of computer- or electronic-based composition, where the fault may be identified as being in a soundfile or synthesis instrument rather than in the score events. There is generally a complex interaction between writing, auditioning and modifying *chunks*, with jumps from one to another and jumps between *chunks* and sub-*chunks* (as well as jumps back to *design framework* or *research* goals).

In order to provide a finished product, the goal *format chunk* is undertaken, normally after a final audition of the *chunk* to be formatted (which may be the entire piece). In fact, up to this point the music may only have existed in the composer's imagination. Formatting can simply involve saving files onto a disk, or could mean laboriously writing by hand a fair copy of a drafted score (and possibly score parts). A formatted version is usually auditioned itself in order to check for any errors that may have been introduced or were not previously discovered. Once all the *chunks* that form the piece have been formatted, then the produce music task is completed and the composition task is generally finished, although naturally it may be reworked subsequently.

3.4. GTM limitations

The goals described above are themselves analysed into further subgoals and procedures, while a concise (but reasonably complete) taxonomy of the objects used in composition tasks has also been developed. There is a common problem for task analysts which is to decide the grain of analysis, i.e. where should the analysis stop in terms of the breakdown of goals into more and more subgoals. In the GTM, the analysis has not always stopped at the same level (unfortunately) and has usually stopped due to one (or more) of three reasons. First, that too little information was obtained from observation/interviews for adequate analysis. In some of these cases, however, we have proposed a speculative structure for a goal that needs to be verified by further experiment. Secondly, that further analysis was deemed too difficult within the scope of the project at the time – i.e. would require new specific investigations to be undertaken. In many of these cases, a cognitive musicological approach may be more suited to further analysis than the more general task analysis approach undertaken so far. Thirdly, that while interesting for general research regarding music composition, further analysis would not be sufficiently relevant to user-interface design purposes. This third reason is the most valid one, in that what should guide the analyst in determining analysis grain is the purpose for which the analysis has been undertaken. While the GTM could be given further depth by future research work, we believe that it indeed does contain sufficient levels of task description to be potentially useful in many areas of music composition research.

A second limitation, other than analysis grain, is that of *secondary* goal relations. Although the model provides a unified structure for the task of music composition, together with necessary ordering of goals, the fact that composers will make various (often opportunistic) jumps within the structure means that the actual patterns of task performance are not directly captured in the model itself. This is to say, *primary* relationships between goals and subgoals are given, but *secondary* goal relationships are not. Examples of *secondary* relations that are likely to be important include those between *design musical structure*, *write chunk* and *design generation systems*. These *secondary* relationships can be very complex and also widely variable between task performers (and performances), making them difficult to encompass within a generalised TKS. Further study of actual task performances, in the light of the GTM, may assist in developing a greater understanding of these relationships and attempts to incorporate them into the model in some way. The absence of such relations can be detrimental to the model's application in user-interface design, since parts of an interface corresponding to areas with *secondary* relations may be left with no links, or inappropriate links, in a resulting system design.

The task analysis itself does not specify directly any part of user-interface to be designed, but rather it provides a framework describing what typical user tasks are, how users interact with the task environment, and an outline of that task environment. There remains a creative leap to be made by the user-interface designer in moving from the TA to an actual user-interface design.

4. THE DEVELOPMENT OF MODALYSER

4.1. Modalyser fundamentals

Modalys – previously known as Mosaic (Morrison and Adrien 1993) – is a physical modelling sound synthesis program based on modal modelling (Fletcher and Rossing 1991). Modalys simulates the vibration patterns that occur in systems of interacting acoustic objects in order to synthesise sounds. The Modalys user-interface works as an extension of the Elk Scheme (Laumann and Bormann 1994) – a Lisp dialect – programming language. Modalys was selected as the target for developing a new user-interface for two main reasons: Modalys is a powerful synthesis

tool that, while capable of signal-based synthesis approaches, principally uses a physical modelling paradigm that should be more easily understood by a wide range of potential users; current users were required to learn Scheme, so provision of a graphical system could help nonprogrammer composers make use of the system. Modalyser (Polfreman 1997a) is a separate application, developed using Digitool's Macintosh Common Lisp, that allows users to create Modalys programs by working in a graphical environment. Modalys-ER represents a futher development, in collaboration with IRCAM, whereby Modalys and Modalyser are merged into a single application – the Modalyser graphical environment communicating with the Modalys synthesis engine implemented as dynamic libraries.

As a part of our research, a questionnaire survey was carried out asking various questions relating to compositional strategies, equipment used, etc. We categorised composers according to the types of technology they used for sound generation during the compositional process (i.e. not for final performance), since it seemed from looking at various systems that different knowledge and expertise is required for the use of these technologies. The classifications were *acoustic*, *electronic* and *computer*. The relative numbers of composers involved in each technology mode (noting that a single composer can use more than one mode) are shown in figure 5. In this figure, using data from a randomly selected sample of composers working in the UK, we can see that acoustic mode is the most common, while computer mode is much lower than acoustic or electronic. Given such an outline, removing the need to write Scheme code seemed to be a distinct priority in the user-interface design, since we could not expect composers not already using computer mode (and not all those working in computer mode) to be familiar with any generic computer programming languages. A necessary adjunct to this was that the system must then enforce syntactic correctness, i.e. given that the system would initially have to work by generating Scheme code for Modalys to execute, it must not allow invalid code to be written.

Given the predominance of the acoustic mode, the use of physical modelling synthesis seemed appropriate, since the aim of the research was to develop a system that could be widely understood by many

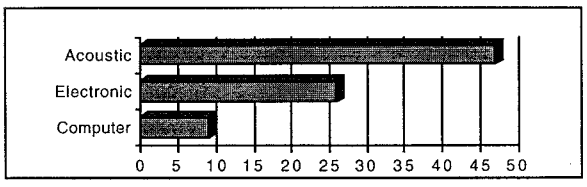


Figure 5. Technology mode prevalence.

composers without previous experience of electronic and computer sound synthesis tools. However, the physical model presented by Modalys, while being very open and flexible, is not immediately straightforward in its terminology and syntax and uses few predefined high-level structures. Figure 6 shows a simple Modalys program that simulates dropping mass onto a tuned plate under gravity (sound example 1).

A Modalys synthesis is specified in terms of *objects*, *connections*, *controllers* and *accesses* (Morrison and Waxman 1991). *Objects* are vibrating items such as plates, tubes and strings. *Connections* define interactions between *objects* and the movements of *objects*. *Controllers* define time-varying (and constant) parameters for controlling a synthesis. *Accesses* define points on *objects* for use in interactions, etc. In figure 6 the first statement is 'new' which clears the virtual workspace to start from scratch. The following two lines of code make two *objects*, a rectangular 'plate' and a 'bi-two-mass' and assign them labels so we can refer to them later. We then define an *access* point on each *object* and make a 'strike' *connection* between them. This means that if the mass moves past the plate, these two points will strike each other. Another *access* point is defined on the bi-two-mass and a 'force' *connection* is made to it in order to apply a downward gravitational force to the mass. One more *access* is defined on the plate and used by a 'make-point-output' in order to turn the vibrations of the plate into sound data. Finally, a 'run' command activates the simulation and a 'play' command plays the resulting sound.

In Modalys there is a single level within which the complete simulation is defined and with no clear separation of performance elements from 'hardware' definitions. However, higher-level organisations of these elements can be user-defined using Scheme's object-oriented programming extension, OOPS. Modalyser uses these object-oriented facilities in the Modalys code it produces. A synthesis in Modalyser is divided into *instruments* and *scores*. The main aim of this is to allow *instruments* to be reused for creating many gestures – the gestural quality of physical models being a key compositional advantage in their use. A second aim is that *instruments* should be able to respond appropriately to any given *score* with only minor adjustments. This second aim is only partially achieved within the current version of Modalyser, due to limitations in its instrument techniques capabilities. The *score/instrument* division is also a familiar conceptual structure for most composers that fits in with the GTM. Figure 7 compares the components of Modalyser with elements from the task model.

We can see from figure 7 both that the division of musical components in Modalyser is slightly different to that of the GTM and that it is also incomplete.

```
;;;-----------------------------------------------------------------------------
;;; A mass dropped onto a tuned rectangular plate, under gravity, no friction.
;;;-----------------------------------------------------------------------------

(new)

(define my-plate (with-pitch-adjustment 'size 440.0
   (make-object 'rect-plate (thickness .001) (density 10000) (freq-loss 0.4)   (const-loss 0.9))))

(define my-mass (make-object 'bi-two-mass (small-mass .6) (large-mass .6)))


(define my-plate-hit (make-access my-plate (const .6 .7) 'normal))
(define my-mass-hit (make-access my-mass (const 1) 'trans0))
(make-connection 'strike my-plate-hit my-mass-hit 0 0.1)


(define my-mass-mov (make-access my-mass (const 0) 'trans0))
(make-connection 'force my-mass-mov  (const -20))


(define my-plate-out (make-access my-plate (const .2 .1) 'normal))
(make-point-output my-plate-out)


(run 10)
(play)
```

Figure 6. A simple Modalys program.

The reason for the difference in divisions is that it was felt that a top-level separation into four separate elements would be too complex in a user-interface, so we opted to divide initially into *instrument* and *score*, but these would then have further subdivisions to match the GTM. The *score* element of the GTM is in fact currently missing from Modalyser. This reflects the fact that Modalyser is still very much in development and a full scoring system for Modalyser is intended as a future component of the software. It is also arguable that considering the slow processing speed of the Modalys synthesis engine, particularly in complex synthesis setups, it is impractical at the moment to synthesise larger-scale *scores*. There is also limited support for generation systems in the current Modalyser release; these also await future development. Despite these and other missing features, we believe that Modalyser can be used effectively for the creation of musical sound gestures.

A Modalyser instrument is divided into *construction* and *techniques* editing areas. The *construction* defines the physical elements of an *instrument*, while the *techniques* represent methods for controlling the *instrument*. Within the *construction* editing area of a Modalyser *instrument*, only *objects* and *connections* are seen (in a patch-type notation). *Controllers* and *accesses* are subsumed into the parameter sets of *connections* – this simplifies the layout and effectively removes *accesses* and *controllers* as separate entities from the conceptual framework, although at the cost of making *connections* more complex. Figure 8 shows the equivalent Modalyser notation to the Modalys program given in figure 6. The result is a very simple-to-read notation that clearly shows the important sound-making elements of an instrument. The downside of this is that parameters are hidden away inside the graphic objects, sometimes several layers deep. Modalyser's construction editor also provides new 'combi' *objects*, which are effectively instrument subpatches, allowing the composer to effectively organise complex instrument constructions containing many objects and connections.

There are three types of instrument *technique* in Modalyser: *pitch*, *excitation* and *timbre*. An *instrument* has one (currently monophonic) pitch *technique*, up to ten excitation *techniques* and as many timbre *techniques* as required. A *score* activates the *techniques* of an *instrument* via three continuous envelope-type editors in order to create a performance.

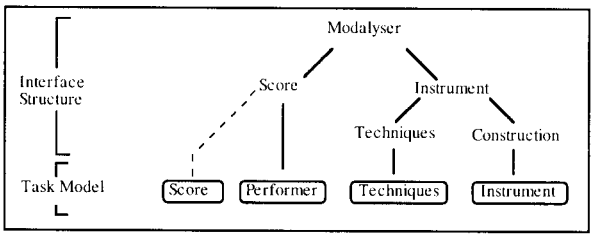


Figure 7. Comparison of the Modalyser and GTM structures.

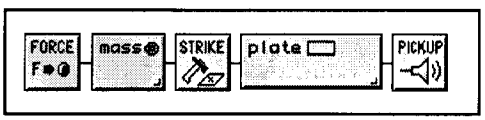


Figure 8. Simple Modalys example – Modalyser equivalent.

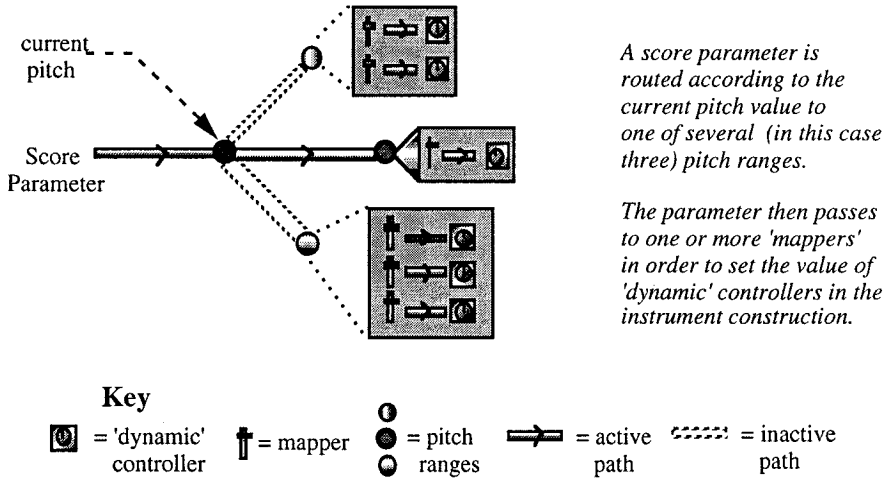


Figure 9. Parameter routing in Modalyser.

The behaviour of a *technique* is determined by a number of *mappers* that translate standardised score values into appropriate Modalys *controller* settings. *Pitch ranges* allow *techniques* (including *pitch* itself) to change behaviours according to the current pitch value, so that, for example, a plucking *technique* could pluck one string within one *pitch range* and a different string in another. Each *technique* has a slot for specifying its behaviour in each *pitch range*. *Pitch ranges* provide a mechanism whereby continuous controls can occur within *pitch ranges* and discrete changes in behaviour can occur at *pitch range* boundaries. The behaviour defined for a particular slot will be activated when the pitch given in the score is within that slot's range. Figure 9 shows an example of routing parameters from a score to an instrument's *controllers*. A score in Modalyser consists mainly of three independent envelope editors that are used for specifying techniques parameters through time – one for pitch, one for excitation and one for timbre.

4.2. A Modalyser example: stringed instrument

Figure 10 shows a stringed *instrument* created in Modalyser. The *instrument* window shows the *construction* (top half) and the *techniques* (bottom half) editing areas. In the top left of the *construction* area can be seen a group of items for bowing the string (position, speed, mass and bow); top central are items for pitching the string (force, mass, fingerboard); bottom central are items for plucking the string (position, mass, pluck), and centre right two pickups attached to the string give a stereo output of the string's vibrations.

In the *techniques* area is only one *pitch range* (far left), covering a two-octave inclusive range. The pitches are shown here as Hertz, but can also be shown as MIDI note numbers. The current *range* selected uses the pitch technique to control the active 'fingerfo Force dynamic' and 'fboard Clamped dynamic'. The first of these applies a force that pushes the 'finger' (a mass object) against a virtual fingerboard, while the second moves the finger to a position along the fingerboard appropriate for giving the correct pitch. Looking at the excitation techniques, we can see that we have 'bow' and 'pluck' techniques and that the bow technique activates two controllers – 'bowpos Position dynamic' (setting the height of the bow relative to that of the string) and 'bowspeed Speed dynamic' (the speed of the bow in a direction perpendicular to the string's length) – see figure 11. The pluck technique controls the 'plectpos Position dynamic' in order to pull the plectrum across

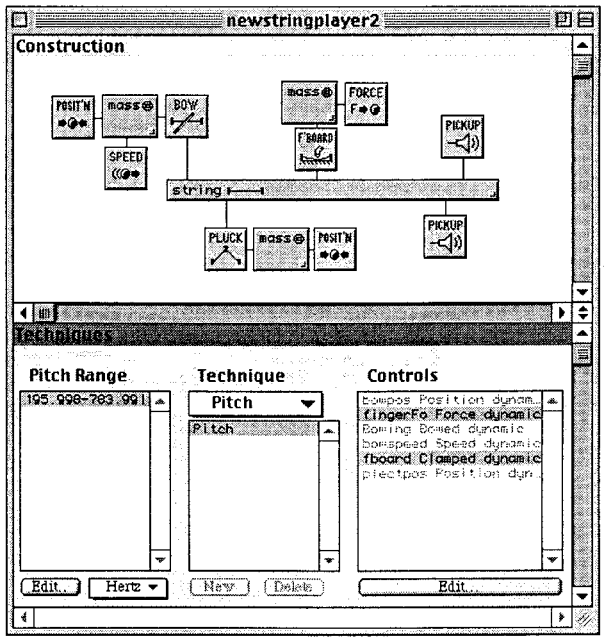


Figure 10. Stringed instrument example.

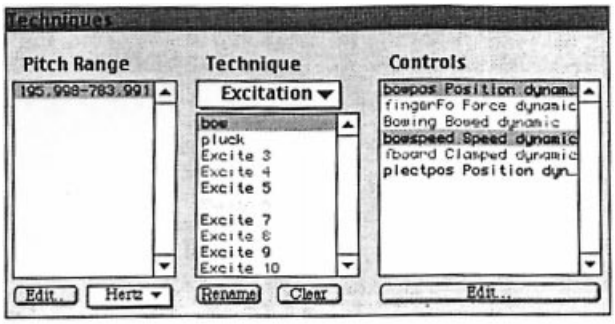


Figure 11. Stringed instrument example bowing technique.

the string, as well as the two bowing controls, moving the bow away from the string and setting the bow speed to zero. The actual values given to controls via a technique according to score variables are set by 'mappers'. These are found in a technique's editing window and currently allow only simple linear behaviours.

Figure 12 shows a part of a *score* used to play this *instrument* (sound example 2). The top envelope is used for setting the *pitch* (currently only in Hz). The horizontal grey lines on the pitch envelope indicate the boundaries between one *pitch range* and the next – here we only have one range. The second envelope controls the *excitation*. This shows one parameter as height on the vertical axis and the other as the density of shading (the parameters can be swapped in order to select which one to edit). In the figure, this is showing *pressure* as the vertical parameter, *movement* as shading. At either end of the envelope, in the areas where pressure is at its maximum value, the instrument is in fact being plucked, whereas in the central region it is being bowed. This indication is given in the form of colours displayed, here red for the plucking and black for the bowing, but this cannot be seen here. In this example, then, the instrument is plucked, then bowed, then plucked again – with various pitch changes during the phrase. Bowing is one of the most difficult interactions in Modalys to control, particularly when interfered with by using a fingerboard and repitching the string this way.

4.3. Modalyser future directions

As stated earlier, Modalyser is not yet complete. 'Complete' here has two meanings: first, in terms of providing as much of Modalys' capabilities as possible and secondly, in terms of enabling the substantial majority of GTM tasks to be carried out within

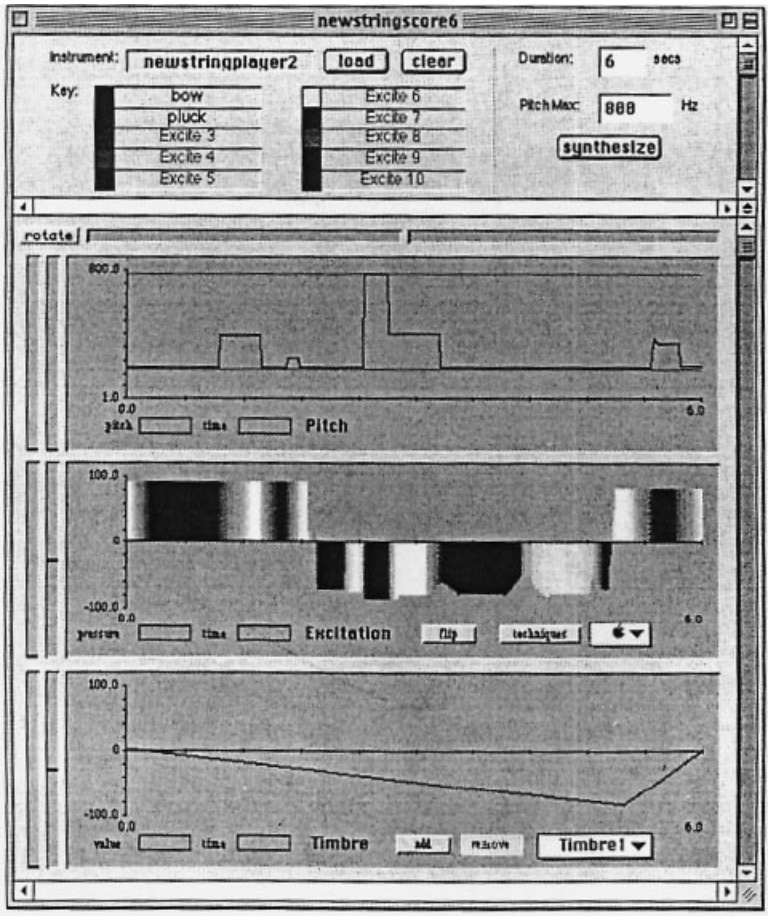


Figure 12. Stringed instrument example score.

Modalyser. In addition, there are various problems with Modalyser in its current form that reduce its usability. These mainly relate to the 'separation' or 'distance' between the user and the virtual instrument they are using. This distance is in both time (Modalys is not realtime – the user must create an instrument, then a score and then synthesise, which itself may take several minutes) and space (the user interacts with an abstract 'patch' description of an instrument's construction; when a synthesis is run there is only sonic feedback regarding what occurred during the simulation).

The current version of Modalyser is written in Macintosh Common Lisp and runs only on Power Macintosh systems. We are currently, in collaboration with IRCAM, investigating several possibilities for the future development of Modalyser:

- The first product of this collaboration has been the forming of a tighter integration of the Modalyser system with Modalys – producing Modalys-ER. This single application version remains to be improved by supporting more of Modalys' features within the graphical user-interface environment. This version has already helped to reduce the 'distance' between the user and the synthesis process. First, the user no longer has to export files manually and then load these into a separate application for synthesis. Secondly, a new 'test' feature has been added to object editors that allows the user to explore the timbral qualities of an object using a rapid synthesis of a mass striking the object at a user-specified location with a user-specified force (see figure 13). The user can easily change the parameters of the object – such as the material it is made from – and quickly resynthesise to hear the effect of the change. Modalys-ER is available to members of the Analysis–Synthesis group of the IRCAM Software Forum.
- In the longer term, the development of a new integrated Modalyser/Modalys application using the Java programming language (for the user-interface) and C++ dynamic libraries (for the synthesis engine) is envisaged. We would redesign some of the Modalys-ER user-interface in order to improve user visualisation of synthesis setups and provide much better user feedback regarding synthesis events, while retaining the structural elements of the current Modalys-ER implementation. This application would also aim to provide support for more GTM features more effectively than in the prototype system. Such an application should be available on multiple platforms (the choice of using a Java/C++ combination is with this portability in mind) – including MacOS, SGI and Linux. In this development we hope to investigate the potential for 3D graphical notations in instrument specification, synthesis feedback and high-level score structures, using the Java 3D API.

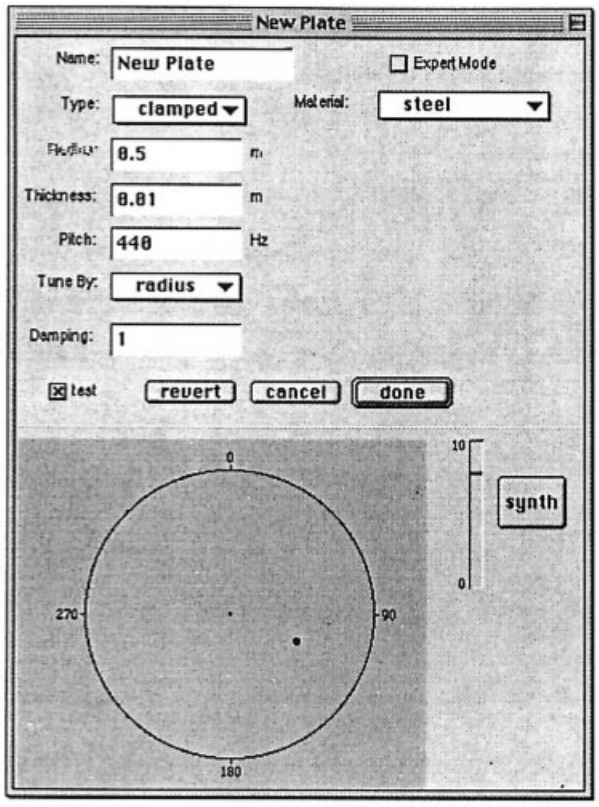
Within these future development paths, various specific changes to Modalys-ER are envisaged:

- extension of technique *mappers* capabilities,
- support for polyphony in scores,
- expansion of 'test' modes to include interactions other than strike (such as pluck, bow),
- higher-level score structures for arranging, relating and ordering musical gestures,
- provision of functions for both generating and transforming score data,
- possible use of AI techniques for providing users with assistance in both instrument construction and performance specification, and
- new interaction possibilities for when realtime synthesis with Modalys becomes widely available.

5. CONCLUSION

The generic task model presented here was found to be a useful tool in assisting the development of Modalyser, particularly in terms of defining the structure and conceptual levels of the interface as presented to the user. However, the GTM is not used in a simple way. The implementation can be seen as a translation of the GTM according to the constraints of the computer operating system (and in this case Modalys), development time, and further user-related considerations determined outside of the model itself. As the prototype develops, the model should also be of use in checking for absent functionality within the area of the GTM that the interface attempts to cover (it is known that much is absent in Modalyser's current state). The GTM can be seen as a map of potential additional areas that may be supported in the future and also may help in designing software that is aimed at fitting in with other computer software packages – by identifying the areas of the GTM that are covered (and hence those that are not covered) by existing computer music software. Future work may involve a detailed review of some extant computer music systems, using the GTM as a framework for evaluation.

Although our work has been directed specifically at the problem of user-interface design for software-based sound synthesis systems, we believe that the GTM may find wider applications in future research. The TA may provide a useful contribution to the understanding of the complex task of music composition that may be explored further by researchers in either purely music-related fields or in interdisciplinary ones such as computer music and cognitive musicology. As the development of Modalyser continues,



The top half of the editor shows the parameters of the plate, while the bottom half shows the testing facility (which 'drops down' when the 'test' checkbox is checked). Here the user can set a location on the circular plate at which it will be struck and can use the slider to set the force with which it will be struck. Pressing the 'synth' button will create the sound.

Figure 13. A circular plate editor with 'test' facility.

it is hoped that the world of powerful software sound synthesis can successfully be made more accessible to nontechnicians and can be presented in a more generally appropriate framework for music composition.

APPENDIX: A LISTING OF THE SOUND EXAMPLES

Example 1. The Modalys example shown in figure 6 (a mass dropped onto a plate).

Example 2. The example in figures 10–12. A string plucked/bowed and repitched using a fingerboard and 'finger'.

Examples 3–4. Further examples using the previous Modalyser instrument with different scores.

Examples 5–6. A reed-tube instrument (clarinet-like), repitched by opening/closing holes.

Examples 7–8. A real trombone note, then that sound resonating a tube in Modalyser.

Examples 9–10. Bowed strings using sinewaves modulating the pickup levels – by composer Robert Wright.

REFERENCES

Eckel, G., Iovino, F., and Causse, R. 1995. Sound synthesis by physical modelling with Modalys. *Proc. of the Int. Symp. on Musical Acoustics*, pp. 478–82. Dourdan, France.

Fletcher, N., and Rossing, T. 1991. *The Physics of Musical Instruments.* New York: Springer-Verlag.

Huron, D. 1992. Design principles in computer-based music representation. In Marsden and Pople (eds.) *Computer Representations and Models in Music*, chap. 1. London: Academic Press.

Johnson, P. 1992. *Human Computer Interaction: Psychology, Task Analysis and Software Engineering.* Maidenhead: McGraw-Hill.

Laske, O. 1992. Artificial intelligence and music: a cornerstone of cognitive musicology. In Balaban, Ebcioglu and Laske (eds.) *Understanding Music with AI*, chap. 1. Cambridge, MA: MIT Press.

Laumann, O., and Bormann, C. 1994. Elk: the extension language kit. *Usenix Computing Systems*, **7**(4). USENIX.

Morrison, J. D., and Adrien, J.-M. 1993. MOSAIC: a framework for modal synthesis. *Computer Music Journal* **17**(1): 45–56. Cambridge, MA: MIT Press.

Morrison, J. D., and Waxman, D. 1991–8. *Modalys Reference Manual.* Forum IRCAM Documentation, IRCAM.

Polfreman, R., and Sapsford-Francis, J. 1995. A human factors approach to computer music systems user-interface design. *Proc. of the 1995 ICMC.* ICMA.

Polfreman, R. 1997a. *Modalyser User Manual.* University of Hertfordshire.

Polfreman, R. 1997b. *User-Interface Design for Software Based Sound Synthesis Systems.* PhD Thesis, University of Hertfordshire.