Design Patterns for Interactive Musical Systems

Jan Borchers and Max Mühlhäuser
Linz University, Austria

We propose Musical Design Patterns as an approach to developing interactive, music-oriented systems for use as novel media content. Such development integrates three key aspects: user interface design, modeling media semantics, and application software engineering (content development). We will support our claims with examples from WorldBeat, an award-winning interactive music exhibit on display at various technology museums.

As Xerox Palo Alto Research Center (PARC) researcher and Apple fellow Alan Kay put it, technologists do not invent new media, but merely new media technologies, and it is the creative authors and content creators who define new media based on those technologies. This is why it usually takes a long time from the invention of a new media technology to the actual emergence of new media types. For example, it took several hundred years from the invention of the printing press to achieve our variety of print media today. It also implies that we're still in the infancy of exploiting those networked multimedia technologies people like to call "new media"—most content presented with those new technologies still uses document metaphors from the printing age.

Interactivity and semantics of new media

One encompassing principle, however, has become a fundamental feature of new media: interactivity. From "interactive" TV to virtual reality environments to telecooperative applications, it is the possibility of interacting with such systems—or with other people through them—in formerly unknown, direct ways that makes these scenarios so attractive.

While the multimedia capabilities of current consumer systems compared to their respective features 10 or 15 years ago might fascinate us, their abilities to let users interact with that data in media-appropriate ways haven't exactly kept up—they're practically nonexistent. Of course, multimedia systems have evolved from the "device handle" (for example, offering VCR-like controls for digital video access) to the "data handle" paradigm. Currently, multimedia systems aim to establish a "contents handle" level of abstraction in the contents model and user interface. Standards being developed to deal with the semantic modeling aspects of this goal include the Moving Picures Expert Group's MPEG-4 for encoding multiple objects within an audio-visual scene and MPEG-7 for describing multimedia information. To create this semantic information, many ambitious research projects currently reconstruct it from digitized data by syntactic analysis, pattern matching, and so on.

However, as research publications in this field indicate, rebuilding semantic information from digital data proves difficult and less satisfactory than storing content information—that is, semantics—at the time of media creation. Moreover, approaches such as syntactic analysis and pattern matching still do not reflect the central role of interaction in future systems. Given a sufficiently pessimistic view, this situation could well be seen as the beginning of a new software crisis.

It is not just "human-computer interaction people" who see things this way. Theoretical computer scientist Peter Wegner[1] even predicts a paradigm shift in computer science as a whole: The algorithm—as the one unifying concept that basically defines computer science—will be replaced by the notion of interaction.

Development needs support

If the computer science community wants to support this paradigm shift, what must we do? Clearly, the key lies in methodical, architectural, and tool-level support for designers and developers of "new media" applications. This means that the following three domains, which often coexist, must be supported by a more unified approach:

Semantic modeling. Without a rich, more abstract model of multimedia information (music provides the perfect example), any attempt at supplying useful and meaningful user interface metaphors will be pointless. The interface cannot represent multimedia data in a semantically rich way if those semantics—that is, the content structure and meaning of the data—are not modeled within the system. We need a model that represents this semantic information, while at the same time incorporating the special demands of interaction and content creation.

Software engineering. Creating interactive multimedia applications is exponentially more

1070-986X/98/$10.00 © 1998 IEEE

complicated than developing traditional, less interactive software. We will need to find an approach that helps developers think about these new types of interactive, networked multimedia systems in a way that suits the nature of this new domain. Object orientation offers a starting point, but we will need to find a larger scale model than that. Otherwise, its architectural, static approach leads even further away from the task-oriented models of user interface design.

Human-computer interaction (HCI). If we take the fundamental role of interaction seriously, then it must be a central part of our approach and not an add-on that "needs to be done to make the system look nice." That's not to say that user interface and application-internal functions should be thrown together. However, we want the HCI and software engineering sides of a project to work together seamlessly, contrary to today's practice. Although studies repeatedly indicate that about 50 percent of the implementation work in software projects go into the user interface,[2] those two camps hardly combined efforts until 1994when they met in a first joint workshop.[3] This workshop stimulated, for instance, a modality-abstracting user interface toolkit and design method, which tightly integrates expertise from both disciplines.[4]

Before we present our approach to solving these problems, we want to give an overview of how the intersection of our two major issues—music and human-computer interaction—has been covered in the past.

Music as interactive medium

Music and human-computer interaction can combine in two fundamental ways. First, computer systems can be enhanced by interactive techniques that use music as a communication medium. Second, a computer system that handles musical data can be improved by finding new and more appropriate interaction methods for it than offered by a computer keyboard and mouse.

A number of research projects have studied the first alternative, although they just scratched the surface of its possibilities. The second alternative we will have to address in our context. Probably because of the music-specific application domain, even less computer science-oriented research has gone into this field up to now.

As an example, Rowe[5] gives a good overview of interactive music systems and classifies existing systems according to three dimensions. He distinguishes score- and performance-driven systems; transformative, generative, or sequenced responses; and instrument- or player-like behavior. While he points out that changes of behavior in response to live input are the hallmark of interactive music systems, his work leaves out actual user interface design systems. Paradiso's[6] recent overview addresses this subject in more detail, albeit focused on the hardware level.

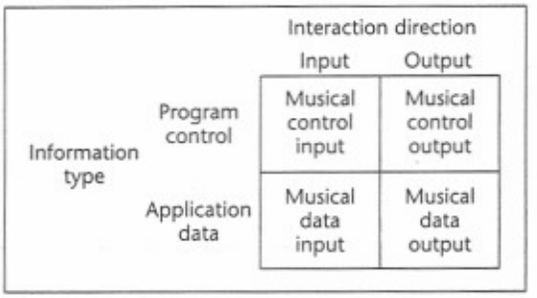

Figure 1. The dimensions of using music as an interactive medium.

Nevertheless, both alternatives we initially identified require "music-appropriate" interaction metaphors, whether they actually use sound to interact or other adequate metaphors (like conducting). Therefore, it's helpful to structure a survey of music in interaction by two dimensions typical of the HCI domain (see Figure 1): first, by the direction in which it is used (input or output), and second, by the type of information transmitted (program control information or application data).

Musical data output

Playing back musical data in auditory form comes as a standard feature on modern personal computers. It's used not only in music applications like software sequencers, but also in games and multimedia "edutainment" software products (see the sidebar "From Digitized Audio to MIDI, and Back").

"Unmusical" domains have shown promising

From Digitized Audio to MIDI, and Back

Early attempts in musical data output used digitized audio information. This corresponded to the lowest level of musical abstraction—physical signals encoded in a binary stream without any meaningful structure. It also put considerable demands on the processing hardware involved.

With the advent of the Musical Instrument Digital Interface (MIDI) standard in the 1980s, this computation-intensive task was quickly moved into specialized sound-generating devices. The CPU sends MIDI messages to the device. The messages contain short commands with meanings like "Play a middle C with 50 percent velocity using a trumpet sound." The sound generator uses stored audio samples or synthesizing circuitry to create the corresponding sound.

Nowadays, personal computers have become powerful enough to do this sound generation in software, which has lead to software sequencers that can convert digitized audio data into an (approximate) MIDI representation and back, effectively blurring the boundaries between MIDI and digitized audio.

results with music as the output medium for scientific data and even algorithm audiolization. Alty et al.[7] give a brief overview of their own and related work elsewhere.

Musical control output

The use of sound to inform the user about program states, to create feedback on commands, and so on, have been studied in more detail. Examples include Earcons, auditory icons introduced by Blattner, Sumikawa, and Greenberg,[8] and the SonicFinder, which uses audio signals for many common operations on the desktop.[9] Actual music, however, has rarely been used to communicate control information. Alty et al.[7] are working on Audiograph, a translator from graphical to auditory user interfaces for the blind.

One factor that limits the usefulness of audio output is that audio remains an undirected medium, contrary to visual media. This leads to unwanted noise whenever several people use such systems while working in a single environment, like a shared office. Nevertheless, unobtrusive and mutable sound effects have made their way into today's standard desktops (for example, a page-turning noise when reducing a window to its titlebar under the Macintosh operating system).

Musical data input

Getting musical data into music applications is also an established feature, either by recording digital audio or by capturing MIDI messages created on an electronic instrument with a MIDI interface. People who cannot play a classical instrument, however, have few ways of bringing their creative musical potential into the computer. Fortunately, many alternative devices—like the infrared batons used by our WorldBeat exhibit (discussed later)—developed in recent years can create MIDI data from almost any type of input. However, those systems are mostly far from being mass-market tools. Moreover, they require intelligent support on the software side to help the user create meaningful musical material.

At this point our approach (and thus, the central topic of this article) comes into focus. We strongly believe that new approaches to musical data input make a fascinating and promising field, for a number of reasons:

1. Musical data input represents musical performance, or the "live" part of music—the bridge between locked-room composition and merely consumptive "couch-potato" listening.
2. Computer-based interaction can open a path to new domains of interactive performance in terms of formerly locked-out performers (musical novices or handicapped people), formerly unfeasible set-ups (participants distributed across the Internet), or formerly unfeasible contents (giving the performer control over unattained levels of complexity).
3. As discussed, such interactive applications are likely to be one of the new, interactive "media" that the Internet and multimedia technologies will bring forth.

Accordingly, we address this problem through our own development efforts described here. One WorldBeat component, for example, lets users hum into a microphone to find the corresponding song in a database—a perfect example of using musical data input for a musical task (and, incidentally, one of the "visions" that the MPEG-7 group mentions in their standardization document).

Musical control input

In general, music is not a very appropriate medium to control program functions. After all, it requires the user to create musical input that will be interpreted as a command. Finding an intuitive mapping for the average user between musical structures and, for example, commands of a text editor will be difficult. Moreover, since most commands require a precise input, humming them would lead to the same problems as speech input does. The input must be classified first, and misinterpretation can be critical.

However, for some domains, especially music, music still makes sense as a control input channel. Most software sequencers, for example, can be triggered by a certain incoming MIDI note to start playing a prerecorded sequence, or any incoming MIDI message can start its MIDI recording function. This lets musicians control basic software functions from their piano keyboard without operating the console keyboard simultaneously. Such instruments also deliver more discrete (pitch) values, which makes command interpretation easier and more reliable.

WorldBeat: An example

In the remainder of this article, we will frequently refer to the WorldBeat system, an interactive computer-based exhibit about computers and music. At this point, we want to provide a brief overview of the system.

Our research group designed and developed the WorldBeat exhibit to show how computers can be used to interact with music in new ways. It's on permanent display at the Ars Electronica Center (AEC), a "technology museum of the future" in Linz, Austria, and is one of the museum's most successful exhibits. WorldBeat received the 1998 Multimedia Transfer Award as one of nine winners from 160 international contestants.

The WorldBeat system puts our idea of Musical Design Patterns into practice. The system's most interesting feature is certainly its user interface: The complete exhibit is controlled using a pair of infrared batons (produced by Buchla and Associates, see Figure 2).

To use the WorldBeat exhibit, a visitor stands in front of the wall-mounted screen and controls the onscreen cursor by pointing with the right baton (see Figure 3). This metaphor resembles that of a laser pointer. To select, say, a menu item on the screen, the user presses the action button on the baton. Once the user has reached one of WorldBeat's components, the screen explains how to use the batons in musical interaction. When finished with that component, the user can use the baton to navigate to a different part of WorldBeat. This dialog technique makes switching between navigational and musical input very intuitive. The user interface is consistent, easy to learn, and adequate for the media-type music.

WorldBeat consists of several components that let visitors interact with music in various ways, such as

- *Joy sticks*. Visitors can play numerous virtual instruments that simulate existing or imaginary instruments. The playing metaphor (mallet-like, strumming, and so on) to use with the batons depends on the instrument selected.

- *Virtual baton*. Visitors can conduct a prerecorded MIDI piece played back by the computer, influencing its tempo and dynamics. The system, based on Lee, Garnett, and Wessel's work,[10] shows how humans can make computers adapt to their own human requirements and metaphors when interacting with them about music.

- *Musical memory*. In an interactive educational game, visitors can try to recognize instruments by their sound alone.

- *Net music*. Visitors can exchange MIDI compositions or even play together with others around the world via the Internet. This scenario shows the use of networks for distributed learning. It will be substantially extended in the future (see the section "Current work").

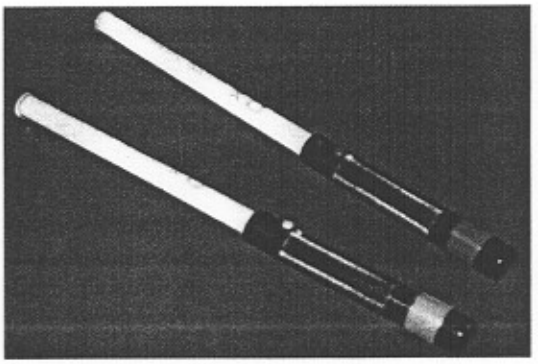

Figure 2. The infrared batons used in the WorldBeat exhibit.

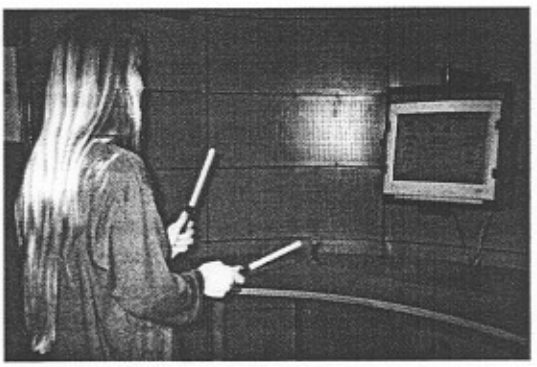

Figure 3. A visitor using the WorldBeat exhibit in the Ars Electronica Center.

- *Query by humming*. Visitors can hum part of a tune to find the corresponding piece in a database.

- *Musical Design Patterns*. Visitors can "customize" the style of a generated blues band, and even improvise to this accompaniment, without having to fear playing the wrong notes.

We will look at some of these components (especially the computer-aided improvisation, of course) in more detail in our description of sample Musical Design Patterns. The baton interface and general design rules for interactive exhibits appear elsewhere.[11]

The Musical Design Patterns approach

Here we'll explain why object orientation offers a suitable paradigm for developing interactive musical systems. Then we'll look at the higher level software engineering concept of Design Patterns, and finally carry this idea over to our problem domain.

Musical objects

A key strength of object orientation is the relatively straightforward path from real-world or conceptual objects to program code in the form of software objects. This makes object orientation a natural approach for representing music in a computer system today. When people think or talk about music in a constructive, technical, or systematic discourse, they primarily deal with musical objects.

On a low level of abstraction, such objects may be notes and rests. But more abstract concepts emerge by composing these simple objects, either horizontally in time (note sequences become a melody) or vertically in voices (simultaneous notes become chords). Phrases and choruses extend the temporal dimension, while timbres and instruments add to the voice dimension, until they comprise a whole musical piece as represented in a score.

Even when the underlying programming paradigm differs, scientists have often adopted the object-oriented approach to model those multiple levels of abstraction (as Haus and Sametti did with "Musical Objects" in their Petri-Net-based ScoreSynth system[12] or Hudak and Berger[13] at Yale University with object nomenclature in their functional Haskore project).

Furthermore, we're looking for an approach that incorporates aspects of interactivity as well as software engineering and modeling—objects interacting via messages suit this approach exceptionally well. That's why practically all modern user interface toolkits are implemented as object-oriented architectures—even when the major application programming interface uses a procedural language like C.

However, just as object orientation alone does not necessarily improve the quality of software, an object-oriented approach to modeling musical concepts does not clearly specify what object classes should be defined and how they should interact in order to handle specific issues of interactive musical systems. We use Design Patterns to build just such a problem-solving vocabulary of concepts.

Design patterns in software engineering

Architect Christopher Alexander[14] originally described the concept of pattern languages to create a vocabulary of proven solutions for recurring urban architecture design problems. Patterns first entered the software development community in 1987, when Tektronix software engineers Ward Cunningham and Kent Beck successfully applied Alexander's ideas to create a small pattern language for novice Smalltalk software designers. They reported their findings at an Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) conference workshop that year.

Software design patterns became popular when Gamma et al. presented their pattern introduction and catalog in 1994.[15] They defined Design Patterns as an approach to capture software engineering experience, based on the object-oriented design paradigm. Each pattern describes a recurring problem of object-oriented software design and offers an abstract solution to it in terms of objects and interfaces between them. In general, a Design Pattern consists of five major parts:

- The *name* of the pattern lets developers identify and talk about this concept easily.
- The *problem context* states when to apply this pattern.
- The *solution* describes the elements and their relationships to solve the problem.
- The *consequences* discuss advantages and disadvantages of this solution to help developers decide whether it suits their specific purpose.
- The *sample uses* show how this pattern has been applied successfully in existing systems.

Let's look at an example of using design patterns. Since our approach tries to address not only software engineering and semantic modeling but also HCI concerns, our example comes from the world of user interface programming.

One of the intriguing concepts introduced by Smalltalk is the Model/View/Controller (MVC) paradigm that separates the user interface and application domain objects from each other. Model objects implement concepts of the application domain, View objects visualize them in the user interface, and Controller objects capture user input for them, managing the interaction. This partitions the object world of an application into these three distinct subsets, making it easier to exchange, for example, the graphical appearance (View) of a piece of software without having to modify its Model or Controller parts.

Most later user interface toolkits, like the recently introduced Swing toolkit (part of the Java Foundation Classes), slightly redefine the partition. Since View and Controller are often too

closely related to decouple, they combine into one partition, reflecting the separation into application-specific and user interface code.

Nevertheless, as Gamma et al. observed, the MVC approach serves as an application of more general concepts, examples of Design Patterns: their Observer pattern captures the concept of allowing different Views to exist simultaneously for a single Model and the principle of notifying them all when the Model changes. Gamma et al.'s Composite pattern describes how Views can be nested, yet still be treated like a single View, and their Strategy pattern reflects the approach to exchange Controllers at any time to activate different algorithms, or strategies, when reacting to user input.

A more detailed discussion of Design Patterns in software engineering exceeds the scope of this article; however, you should now have a sufficiently clear idea of their basic concepts to transfer them to interactive musical systems.

Musical Design Patterns

Recently, patterns have been formulated for other domains, such as software project management. Those "organizational" patterns capture best practices in managing all aspects of the software engineering process. Also, Gamma et al. admit that domain-specific patterns must be created. With Musical Design Patterns, we now take this idea back from pure software engineering into the interdisciplinary field of designing object-oriented, interactive, and computer-based musical systems. We define a Musical Design Pattern as one that describes how to solve a certain problem when designing computer-based, interactive music-oriented systems. It consists of

- a *name* that captures the goal of the pattern,
- a *problem statement* describing a certain abstract musical feature or concept that a system should implement,
- a *solution* as a set of objects taking over different roles and a set of relations between those objects,
- the *consequences* that give hints on when to apply this pattern and how it may influence an overall design, and
- some *sample uses* that show how the pattern has been applied in existing computer-based interactive music systems.

It's important to see that a Musical Design Pattern should not just represent a single, concrete musical concept (like a certain style to play a bass line in jazz music). Instead, it should be an abstraction from this level and capture the general problem to reflect a certain aspect of musical composition, performance, and improvisation in a system design so that applying this pattern creates specific implementations for various stylistic and technical facets of this concept.

Sample patterns

To give a better understanding of Musical Design Patterns, we'll present two examples. The first pattern, MetricTransformer, looks at the rhythmic dimension of music. It captures common structures of small changes in rhythm that characterize real, performed music. The second pattern, ImprovisationHelper, deals with the harmonic dimension. It models different variations of subtle computer support for both novice and experienced human performers who play together with just the computer, via the computer, or with other people.

While a full discussion of these patterns with all implementation details would exceed the scope of this article, we'll present the key concepts, components, and sample uses of each pattern, and evaluate their usefulness in our three problem domains—semantic modeling, software engineering, and HCI.

MetricTransformer pattern

This pattern addresses the rhythmic dimension of music. We'll present it according to our definition of Musical Design Patterns.

Problem. Musical performance adds countless subtle variations to the lifeless representation of a piece in a musical score or simple MIDI file. While these variations concern all musical dimensions—harmony, melody, and rhythm—the one-dimensionality of time makes rhythm the most accessible concept for computer modeling.

This pattern aims to describe which small temporal changes within each beat make musical performance more human and vivid. Examples of these changes are the groove in jazz music (see below), the statistical deviations from the exact beat timing typical of natural performance, or the capability of a real band to follow a human rhythm even if not uniformly spaced.

Solution. The general approach to modeling

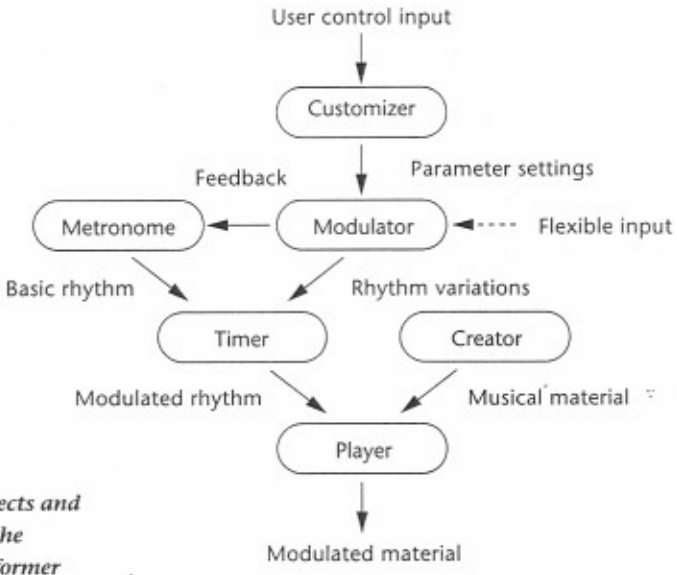

Figure 4. Objects and messages in the MetricTransformer pattern.

this behavior builds on the collaboration of six objects (see Figure 4):

- The Creator supplies the musical "raw material," that is, the score to be played.
- The Metronome supplies the "raw rhythm," that is, the uniform beat at the general tempo of the piece.
- The Modulator defines variations of any basic rhythm in terms of the deviations that form rhythmic "spacing." The modulation can be based on fixed, random, or human deviations. The semantic concept of metric transformation is modeled here. In cases where gradual tempo changes also affect the future tempo as a whole, the Modulator can feed these changes back to the Metronome.
- The Customizer lets the user change the Modulator's parameters in real time. It defines the interaction metaphor when varying the metric transformation.
- The Timer takes the basic beat from the Metronome and modifies it according to the Modulator's input. Its output is the modulated beat that this pattern is supposed to model.
- The Player takes the musical material from the Creator and the modulated beat from the Timer. It outputs the material in this rhythmic form, still in the shape of musical symbolic events (comparable to notes).

Some of these objects contain architectural features that also have been identified as general patterns in the literature. For example, introducing the dedicated Modulator object shields the other objects from changes in the method used to vary the timing. This concept follows Gamma et al.'s Strategy design pattern[15] mentioned earlier.

Consequences. If the Modulator is built to move some notes back in time (that is, to an earlier point than the t_0 scheduled by the Metronome) by up to δt_{max}, then the time of the next beat and the material to play there must be known in advance (at the latest at time $t_0 - \delta t_{max}$). Thus, if the musical input arrives live from a human performer, the system cannot apply this pattern and still deliver its part simultaneously to the input.

Sample uses. The MetricTransformer pattern can be used to model different aspects of timing deviations within a beat from a fixed, uniform tempo as written in a score. The so-called groove in jazz is a good example to apply the MetricTransformer: Often, a band's rhythm section shifts the intermediate beat back in time in a triolic fashion (that is, by one third of its length) to make the music "swing." For example, if a drummer's score contains a sequence of eighth notes to play on the Hi-Hat, the drummer will interpret them roughly as an alternating sequence of 2 + 1 triolic eighth notes (which really have a length of twelfth notes) to create the swing feeling. Comparative studies of jazz recordings show that most bands actually have their own specific groove timing, helping us recognize the band.

The Creator can use prerecorded musical material or quantized (metrically straightened) input from a performer to add groove. The Modulator can then model this concept with a delay algorithm. The delay can be defined as the percentage of time between two beats at which the intermediate beat occurs. For example, 50 percent equals a straight, march-like style, and 67 percent results in a triolic swing. Timings behind that create an even more "laid-back" style used in slow blues, while timings less than 50 percent give an unusual, driving feeling in the music. The Customizer acts as a slider to let users choose this percentage while the music plays, so they can hear the results immediately. Our WorldBeat system, for example, features such a slider onscreen. Current electronic instruments like keyboards, on the other hand, do not have anything similar to offer.

Simply delaying all note events that are scheduled for the time of the intermediate beat and afterwards by a fixed amount, however, is unsatisfying because it lowers the average tempo. Plus, a note that ends shortly before the intermediate beat, for example, remains unchanged, creating a noticeably larger break between it and the next delayed note. The correct approach is to expand (slow down) the complete time line linearly up to the intermediate beat and compress it (speed it up) from the intermediate on. Spreading out the delay over the entire beat this way, even scores with note events that arrive slightly before the intermediate beat—or at any other moment—will sound correct.

The MetricTransformer could also model the jitter, that is, the slight unsteadiness in timing inherent to natural musical performance. Computer-generated rhythms otherwise sound very artificial and lifeless. The Modulator can create this jitter using a statistical standard distribution of time deviations from the standard beat, although more sophisticated models are possible. The Customizer can offer users one or more sliders to change the distribution parameters interactively. In this case, some notes would be moved to an earlier point, so simultaneous real-time input (from the Metronome) and output (by the Player to the Realizer) are not possible, as previously discussed.

Finally, if the system does not use its internal timing but follows the human rhythm of a conductor or band leader, then the Modulator would be implemented as a component that detects the beats in some human input. The Metronome would just define the basic tempo at the beginning. The Customizer would offer settings to users, like the speed with which the computer adjusts to a sudden change of tempo by the conductor, from "immediately" (possibly dropping notes in case of a tempo increase) to "gradually" (spreading out the change over an adjustable time interval). WorldBeat offers a similar scenario in its Virtual Baton component where users can control tempo and dynamics of a piece by conducting with an infrared baton.

Evaluation. From the semantic modeling perspective, the MetricTransformer pattern offers an approach to capture various types of human factors in rhythmical performance in a single framework. It can represent "knowledge" about how to enrich the concept of rhythm in a musical system, using mathematical definitions as well as external human input to describe subtle deviations from a uniform beat.

For software engineering, the pattern shows how to design an interactive music system that deals with this richer concept of rhythm in an integrated way. The model is concentrated and hidden inside the Modulator object, making it easy to add new related aspects of rhythmic variation to a system. Data input can be taken from a stored musical representation or a live performance using appropriate electronic instrument interfaces. This design can represent "interactive media"—at all necessary points, algorithmically generated, stored, or spontaneous human input sources can be plugged in.

Finally, this pattern addresses HCI concerns because the Customizer, while a separate object for architectural clarity, is nevertheless closely connected to the Modulator (the concept representation). This allows for the implementation of richer, media-adequate interface metaphors so that users can interact with the rhythmic concepts in a more meaningful way.

ImprovisationHelper pattern

This second pattern example will look at the harmonic dimension of music.

Problem. An interactive music system should not mimic traditional instruments, but offer new conceptual instruments with semantically rich, interactive features not found in existing equipment. One of the most interesting new features possible is intelligent player support—the system lets users play the creative part, but supports them with its musical knowledge, correcting small errors automatically.

However, to avoid stifling creativity, the system should not influence or "correct" the player in more than one musical dimension. Since melodic or especially rhythmic corrections quickly lead to a loss of perceived immediacy for the player, the system should focus on a harmonic improvisation support. In other words, it should let users play notes rhythmically free (play notes whenever they wish) and melodically free (play high or low notes, runs, chords, and so on), but "fine-tune" the notes played so that they fit into the current harmonic context of the accompaniment.

Such improvisation support must be done carefully to not limit users' creative freedom. Ideally, it would adapt to their level of expertise.

Solution. The ImprovisationHelper pattern consists of the following five objects (see Figure 5):

- The Accompanist supplies the musical accompaniment, which can come from different sources (see sample uses).
- The HarmonicAnalyzer constantly determines the current harmonic context (base and modus) delivered by the Accompanist, in real time.
- The InputAnalyzer offers users a musical interface to play their part in real time and reads their input, such as an improvisation to the accompaniment.
- The Corrector takes the "raw" musical input from the user and the current harmonic situation from the HarmonicAnalyzer, and adjusts the user input so that it fits into this harmonic context. The Corrector's output is a harmonically checked version of the original material played by the user.
- The SupportAdaptor offers users a controller to decide how much improvisation support they want. Experienced players can lower this to get more creative freedom (and responsibility for any wrong notes, of course).

Consequences. The pattern makes a system simulate an "intelligent instrument." As such, interactivity proves especially crucial in this pattern. If the delay between musical user input and corrected system output becomes noticeable (more than about 150 ms), users will lose the feeling of playing an instrument and start thinking that they merely control some artificial music generator.

Also, the user interface should be designed so that the input requested from users is not too exact, but rather slightly fuzzy (for example, the system should not demand input from an 88-key professional piano keyboard). This helps make the system's corrections less noticeable.

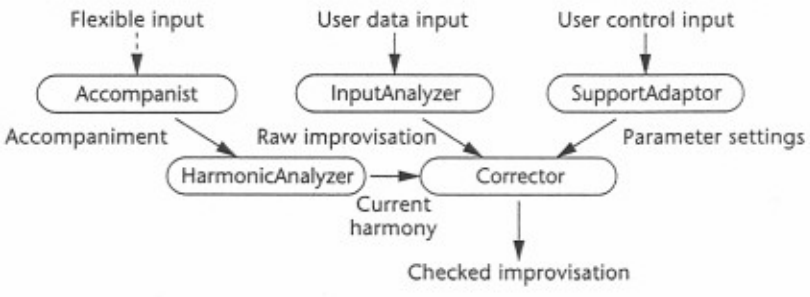

Figure 5. Objects and messages in the ImprovisationHelper pattern.

Sample uses. In the WorldBeat system, we implemented such an improvisation support in the Musical Design Patterns component. The Accompanist supplies the computer-generated accompaniment of a blues band (whose groove can be adjusted using other patterns like the MetricTransformer described above). The HarmonicAnalyzer uses a root-parsing algorithm as described in standard music literature to determine the current chord (say, Fm^7, the F minor 7th chord) in real time. The InputAnalyzer offers a xylophone-like playing metaphor—users make downbeat gestures with the two infrared batons of the WorldBeat system in their hands. Gesture velocity determines volume, while horizontal position determines pitch. The Corrector takes this input and maps it to the nearest harmonically sound note in terms of the current accompaniment chord determined by the HarmonicAnalyzer.

The result is quite fascinating. People who have never before played an instrument can walk up to the system and start improvising to a blues band—without playing wrong notes. This makes the Musical Design Patterns component very attractive and popular among visitors who use the WorldBeat system. Experienced musicians can adjust the support to either a moderately hard-to-play chromatic virtual xylophone using the batons (one octave), or they can use a full-scale electronic piano keyboard that can be added to the system at any time (seven octaves).

The ImprovisationHelper pattern has also been applied independently in MusiKalscope,[16] a system that lets users improvise to a be-bop style of jazz music and control graphical feedback simultaneously. Users play a virtual drum pad using spatial trackers attached to their arms. The RhyMe subsytem analyzes the input and maps it to notes of a matching scale, depending on the accompaniment's current mode. This mode, however, is determined in a separate a priori analysis process that limits the system's flexibility in terms of real-time accompaniment input. The system also lacks some reactiveness due to the input technology and metaphors used. Rowe[5] uses a similar approach to define a "Harmonizer" module.

Another application of the ImprovisationHelper pattern is our NetMusic scenario (see the section "Current work"). Here, humans generate the accompaniment, rather than a computer. The real-time HarmonicAnalyzer ensures that even

this situation, where the harmonic progressions may not be known beforehand, can be covered using the ImprovisationHelper pattern.

Evaluation. For semantic modeling, this pattern is very valuable. It shows how a system can abstract from single notes and let users deal with higher level concepts like harmonic progressions, scales, and so on.

For software engineering, this pattern demonstrates an interesting design: the HarmonicAnalyzer provides a class that does not depend on prerecorded data to accomplish its task. This means that the system can use stored or live accompaniment if this pattern is applied. Again, this is a fundamental accomplishment necessary for software architectures of interactive new media—the data source, whether stored, computed, or provided in real time by a human, can be plugged into the system as desired.

HCI finally is rewarded with an exceptionally intriguing user interface metaphor that creates the impression of an intelligent instrument. Users can actually try out and influence the high-level semantic concepts inside the system, adjusting parameters like the level of improvisation support, and then send the data (their own improvisation) into the system to hear the results.

Current work

Currently, we're working on three aspects of extending the Musical Design Patterns approach. First, we're looking for additional applications of our existing patterns to further prove their general validity. This includes applying them to completely different musical styles. For example, it will be interesting to see how the MetricTransformer can model subtle rhythmic variations—a crucial part of musical expressiveness in performance—in classical music. The ImprovisationHelper is also applied to other styles, such as helping users play a classical minuet.

The second direction of research tries to identify new patterns (the melodic dimension has been neglected in our patterns so far). We're developing a pattern that may capture melodic concepts like musical ornaments (sliding notes or appogiatura in jazz, trills in classical music, and so on). With respect to novel interactive media, user interaction might, for instance, adjust how intensely these effects are added to a performance.

Possibly the single most important extension under way, however, is integrating our existing patterns into a concept of distributed musical collaboration, which will become the next version of our NetMusic WorldBeat component. Extensive theoretical work in this area done already at our Telecooperation department has helped us design a collaborative concept that will circumvent the delay problems inherent in simultaneous real-time cooperative playing via the Internet.

Each participant takes over an instrument, as in a real band or orchestra, and the system distributes basic information about the piece to be created (comparable to a lead sheet). Each player records their own performance locally, which is then sent automatically as a "first take" to the others. That way, all players get a complete first version of the piece with all instruments. Next, each participant can improve and adapt their performance and have it redistributed, until this iterative process leads to the final version of the piece. Even though the players never have to play at exactly the same time (avoiding delay problems), the musical exchange takes place in a tight loop, leading to a highly cooperative experience. We'll have to carefully design the user interface to hide much of the complexity added through the distribution so that NetMusic remains usable with a simple interface like the infrared batons.

To extend this concept to an even more versatile global group performance scenario, we will make the level of synchronicity adaptable to support different situations. Where bandwidth and delay allow (for example, within a local area network or using an ISDN connection), playback could be done simultaneously, and a live video link would increase the feeling of collaboration. If, on the other hand, participants are located in, say Canada, Austria, and Korea, it would make sense to let them enter their tracks during convenient hours in their respective time zones. In that case, we will have to take additional care to preserve the "group feeling." For example, we could record a video of each performer's interactive track creation, then distribute it to the other participants as they use that track.

The key idea, however, is that each player will not have to play their track without support. Instead, our existing patterns will be available to help them create their part if they wish. This means that existing patterns must be applied, for example, to the different instruments and roles in a jazz band. That way, both experienced and inexperienced musicians can create musical recordings (even together in one session) with an individually adaptable level of computer support for each participant.

Conclusion

To develop systems that turn interactive new media into a usable reality, software engineers, modelers of multimedia semantics, and HCI designers need to work much more closely together, and with a new level of support.

Our Musical Design Patterns represent a step toward such an integrated approach, geared to the media type "music." They can be considered a next level of abstraction beyond the several existing object-oriented approaches to musical systems and an attempt to take the importance of interaction seriously.

Sample patterns like MetricTransformer and ImprovisationHelper show what Musical Design Patterns may look like, and the WorldBeat exhibit shows their applicability in real-world systems. Distributed scenarios will add another dimension to their use. Musical Design Patterns thus capture the various domains of experience in designing interactive music systems: how to model certain musical aspects, how to engineer them into a working system, and how to create intuitive, media-adequate user interface metaphors that make such a system actually usable, and thereby useful. **MM**

References

1. P. Wegner, "Why Interaction Is More Powerful Than Algorithms," *Comm. of the ACM*, Vol. 40, No. 5, 1997, pp. 81-91.
2. B.A. Myers and M.B. Rosson, "Survey on User Interface Programming," *Proc. CHI 92*, ACM Press, New York, 1992, pp. 195-202.
3. R. Taylor and J. Coutaz, eds., *Software Engineering and Human-Computer Interaction*, Springer-Verlag, New York, 1994.
4. H.W. Gellersen, "Software Engineering Meets Human-Computer Interaction: Integrating User Interface Design in an Object-Oriented Methodology," *Proc. SOFSEM 95: Theory and Practice of Informatics*, Springer-Verlag, Berlin, 1995, pp. 375-378.
5. R. Rowe, *Interactive Music Systems: Machine Listening and Composing*, MIT Press, Cambridge, Mass., 1993.
6. J. Paradiso, "Electronic Music: New Ways to Play," *IEEE Spectrum*, Dec. 1997, pp. 18-30.
7. J. Alty, D. Rigas, and P. Vickers, "Using Music as a Communication Medium," *CHI 97 Extended Abstracts*, ACM Press, New York, 1997, pp. 30-31.
8. M.M. Blattner, D.A. Sumikawa, and R.M. Greenberg, "Earcons and Icons:Their Structure and Common Design Principles," *Human-Computer Interaction*, Vol. 4, No. 1, 1989, pp. 11-44.
9. W. Gaver, "The Sonic Finder: An Interface that Uses Auditory Icons," *Human-Computer Interaction*, Vol. 4, No. 1, 1989.
10. M. Lee, G. Garnett, and D. Wessel, "An Adaptive Conductor Follower," *Proc. Int'l Computer Music Conf. 92*, Int'l Computer Music Assoc., San Francisco, pp. 454-455, 1992.
11. J. Borchers, "WorldBeat: Designing a Baton-Based Interface for an Interactive Music Exhibit," *Proc. CHI 97*, ACM Press, New York, 1997, pp. 131-138.
12. G. Haus and A. Sametti, "ScoreSynth: A System for the Synthesis of Music Scores Based on Petri Nets and a Music Algebra," *Readings in Computer-Generated Music*, D. Baggi, ed., IEEE CS Press, Los Alamitos, Calif., 1992, pp. 53-77.
13. P. Hudak et al., "Haskore Music Notation—An Algebra of Music," *J. Functional Programming*, Vol. 6, No. 3, Cambridge Univ. Press, UK, 1996, pp. 465-483.
14. C. Alexander, *A Pattern Language: Towns, Buildings, Construction*, Oxford Univ. Press, UK, 1977.
15. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1994.
16. S. Fels, K. Nishimoto, and K. Mase, "MusiKalscope: A Graphical Musical Instrument," *Proc. ICMCS 97*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 55-62.

Max Mühlhäuser is a full professor of computer science and head of the Telecooperation Research Group at Linz University, Austria. His research domain is software engineering for multimedia and multimodal interfaces, mobile and ubiquitous computing, and human-human and human-computer cooperation. He received his diploma and PhD in computer science from Karlsruhe University, Germany, in 1981 and 1986, respectively. He is a member of ACM, the German Computer Society, and IEEE.

Jan Borchers is a research assistant in the Telecooperation Research Group at Linz University, Austria. His research interests lie in human-computer interaction design for new media. He developed WorldBeat and is currently finishing his PhD thesis on interactive musical systems. He received his diploma in computer science from Karlsruhe University, Germany, in 1995 and is a member of ACM SIGCHI and the German Computer Society.

Readers may contact the authors at the Telecooperation Research Group, Dept. of Computer Science, Linz University, Altenberger Str. 69, 4040 Linz, Austria, e-mail {max, jan}@tk.uni-linz.ac.at.