Designing Interactive Music Systems: A Pattern Approach

Jan O. Borchers
Telecooperation Research Group
University of Linz, 4040 Linz, Austria
http://www.tk.uni-linz.ac.at/~jan/

1 INTRODUCTION

Music is a structurally very complex type of multimedia information. Today, computer systems easily play back music in high quality, but an ideal system should offer many more, media-appropriate ways to interact with musical data. Users should be able to hum a melody to locate the tune they want, then conduct it, or play to it with adjustable computer support, alone or with others via the Internet. Such interactivity would encourage musical creativity, and let users learn about musical concepts by playing with them. This paper shows how we used a pattern language approach for software design, interaction design, and the application domain "music" to build a system with these qualities.

2 THE PATTERN APPROACH

Designing such a system requires expertise from software engineering, user interface design, and music to be brought together. We used *pattern languages* to capture and describe this knowledge for all three domains in a uniform way. Pattern languages were originally developed to capture proven solutions to common design problems in urban architecture (Alexander et al. 1977, 1979). Each pattern captures one such "guideline"; the pattern language connects them into a structure, helping the designer to create an overall design of high quality.

The idea has been adapted successfully by software engineering (Coplien & Schmidt 1995), and is currently beginning to find its way into human-computer interaction (HCI) design (Bayle et al. 1997, Tidwell 1998).

2.1 A Formal Pattern Language Definition

To define the components of a pattern language regardless of the problem domain it addresses, we introduce a formal notation.

A *pattern language* is a directed acyclic graph. Each node represents a *pattern.* There is a directed edge from pattern p_1 to p_2 if p_1 *recruits* p_2 to complete its solution. Edges pointing *away* from a pattern are its *consequences,* showing what lower-level patterns need to be applied next. Edges pointing *to* a pattern are its *context,* the situations in which it can be applied. This relationship establishes a *hierarchy* within the pattern language. It leads the designer from patterns addressing large-scale design issues, to patterns about small details.

Each *pattern* is a set $p=\{n, f_1...f_i, s, e_1...e_j\}$ of a *name n, forces* $f_1...f_i$, the *solution s,* and *examples* $e_1...e_j$. It describes a commonly encountered design problem, and suggests a solution that has proven useful in this situation.

While this formal notation helps to clarify the structure of patterns and pattern languages, patterns are actually written texts, to make them easy to read and understand even for people from other professions. Each part of a pattern, and its connections to other patterns, are usually presented as several paragraphs in the pattern description (Alexander et al. 1977).

The *name* of a pattern helps to reference it easily, communicate its central idea quickly, and build a vocabulary. The *forces* are aspects of the design context that need to be optimised. They usually come in pairs that contradict each other. The *solution* describes a proven way to balance these forces optimally for the given context. The *examples* show that the solution has been applied successfully in existing designs. For a more detailed description of the concept of pattern languages, see (Alexander et al. 1977, Alexander 1979).

2.2 Software, HCI, and Musical Design Pattern Languages

A key idea of this paper is to use the pattern approach not only in software or interaction design in isolation, but to structure the problem domain, in this case "music", into a pattern language as well. The rationale behind this approach is that design patterns can be defined for any discipline where some creative design work takes place. In music, the "designer" is a composer or player. Patterns can describe important aspects of how a composition is crafted, or how choices between alternative chord sequences, voices, or single notes are made.

As in the other disciplines, it is obvious that the knowledge captured in a pattern language cannot represent the intuition and creativity of a design expert. But it *is* possible to communicate basic principles of making good designs, or good music, using the pattern approach. Casting the expertise of all disciplines involved into patterns (Fig. 1, only selected patterns shown) has two advantages:

For the development team: By studying each other's pattern languages, it becomes easier for the different professions in the interdisciplinary development team to understand each other's design principles, paradigms, and professional values. The common vocabulary of design patterns from the various professions facilitates communication within the design team.

For the user: Architecture and software engineering have shown that pattern languages are a very suitable format for communicating expertise. An interactive system such as the exhibit we had in mind can use patterns to didactically structure its presentation of musical concepts. These musical patterns can be embodied in user interface objects and relationships that users can see and interact with. The musical concepts become easier to understand.

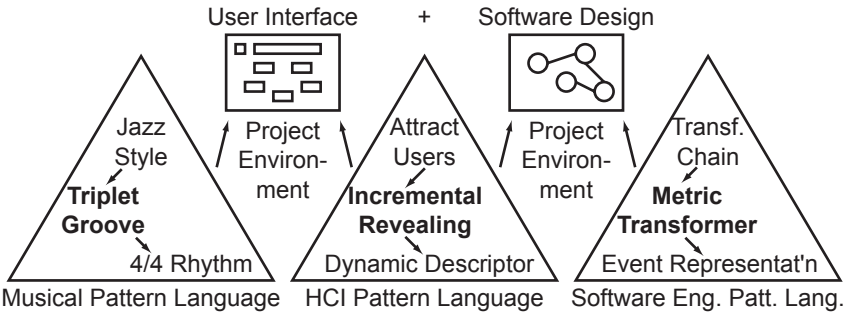
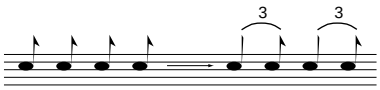


Fig. 1: Design pattern languages for Music, HCI, and software engineering are used in a project environment "interactive exhibit" to create a user interface and software design.

3 PATTERN EXAMPLES

To demonstrate the use of patterns, we will give examples from each domain involved. They have been shortened, and references to other patterns (in italics) left unresolved, to fit the format of this paper.

3.1 Musical Design Pattern: *Triplet Groove*



Context: Playing music in the *Jazz Style.*

Forces: Players need to create a swinging feeling that the straight rhythm from other musical styles does not convey. **But:** Sheet music cannot include all rhythmic variances; it would become unreadable.

Solution: Where the score contains an evenly spaced pattern of eighth notes, shift every second eighth note backwards in time by about one third of its length, shorten it accordingly, and make the preceding eighth note one third longer. The length ratio changes from 1:1 to 2/3:1/3. Two straight eighth notes become a triplet quarter and eighth note. The result is a "laid-back groove".

Examples: Any recorded Jazz piece features this rhythmic shift. The actual shift percentage varies widely: usually, the faster a piece, the less shifting takes place.

Consequences: This pattern uses an underlying straight beat like *4/4 Rhythm.*

3.2 Interaction Design Pattern: *Incremental Revealing*

Context: Decide how to unfold contents and features of an interactive system so that it conveys a *Simple Impression* to *Attract Users,* and *Engages Users*.

Forces: A simple impression is important to make a system look non-intimidating and inviting, especially for novices. **But:** To keep users engaged, the system needs to convey its depth of features and contents as well.

Solution: Initially, only present a concise and simple overview of the system's functionality. When the user actively shows interest in a certain part of this overview, offer additional information about it, revealing in successive stages what lies behind the initial presentation.

Examples: Desktop GUIs hide menu entries in menu bars until the user selects a menu. *WorldBeat* has a simple main selection screen with only names and icons for composing, conducting, etc.; when the cursor is over an item, a short text explains it; when it is selected, the system switches to the new page.

Consequences: Incremental revealing is easier when the contents have a *Flat & Narrow Tree* structure. To show what lies behind a user interface object, use *Dynamic Descriptors* (as in Mac OS Balloon Help, or Windows ToolTips).

3.3 Software Design Pattern: *Metric Transformer*

This pattern addresses the problem that, in a *Transformer Chain* of modules that modify an incoming stream of music in *Event Representation,* one module needs to modify the timing, for example, to implement a component that allows the user to apply *Triplet Groove* to music that is being played. This pattern has been described in detail in (Borchers & Mühlhäuser 1998).

4 THE WORLDBEAT EXHIBIT

WorldBeat is an interactive music exhibit that the author designed for the *Ars Electronica Center (AEC),* a technology "museum of the future" in Linz, Austria. *WorldBeat* offers visitors new ways of interacting with music. Using just a pair of infrared batons, they navigate through the system, play virtual instruments like drums or a guitar, conduct a computer orchestra playing a classical piece, and improvise to a Blues band with computer support. Users can locate tunes by humming their melody, and try to recognize instruments by their sound. Details can be found in (Borchers 1997).

Many design aspects of *WorldBeat* are applications of the musical, HCI, and software engineering patterns, as described in the examples. The success of *WorldBeat* indicates the validity of our approach: AEC visitors rated it among the top three of the center's exhibits, and the system received the *1998 Multimedia Transfer Award* among over 150 international contestants.

We are currently designing a new exhibit to let people compose in the twelve-tone style of Arnold Schönberg. We have successfully cast many aspects of his composition theory into a pattern language. Also, many HCI and software engineering patterns we used in WorldBeat carried over very well to this new project, and helped to communicate experience to new design team members.

5 FURTHER RESEARCH

Our pattern languages have to be refined, extended, and validated further. The approach needs to be applied to different application domains, and our formal notation suggests computer support for working with pattern languages. The author will use the pattern format to teach HCI design to students, to see how novices benefit from this approach.

6 REFERENCES

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. & Angel, S. (1977). *A Pattern Language: Towns, Buildings, Construction.* New York: Oxford University Press.

Alexander, C. (1979). *The Timeless Way of Building.* Oxford University Press.

Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., Gross, B., Lehder, D., Marmolin, H., Moore, B., Potts, C., Skousen, G. & Thomas, J. (1998). "Putting it all together: Towards a pattern language for interaction design". *SIGCHI Bull.* **30**(1), 17–23. New York: ACM.

Borchers, J. (1997). WorldBeat: Designing a Baton-Based Interface for an Interactive Music Exhibit. *Proc. CHI'97* (Atlanta, GA), 131–138. New York: ACM. See also the video proceedings of that conference.

Borchers, J. & Mühlhäuser, M. (1998). Design Patterns for Interactive Musical Systems. *IEEE MultiMedia* **5**(3), 36–46. Los Alamitos: IEEE Computer Society.

Coplien, J. & Schmidt, D. (1995). *Pattern Languages of Program Design.* Software Patterns Series. Reading, MA: Addison-Wesley.

Tidwell, J. (1998). Interaction Design Patterns. *PLoP'98 Conf. on Pattern Languages of Programming* (Monticello, IL). Presentation; extended version at http://www.mit.edu/~jtidwell/interaction_patterns.html.