Breaking the Interdisciplinary Limits of Computer-Human Interaction Design: A Pattern Approach

Jan O. Borchers
Telecooperation Research Group
Linz University
4040 Linz, Austria
+43 732 2468 9888
jan@tk.uni-linz.ac.at

ABSTRACT

A major limit for CHI is communication in interdisciplinary design teams. We propose a pattern-based approach to break this limit. A pattern language that captures experience and values from software engineering, HCI, and the application domain can improve communication and acceptance within the design team, and lead to better products and design rationales. We developed an award-winning interactive music exhibit, *WorldBeat,* and, from our experiences, started to build such a language. We are now using this language to improve the design process of a subsequent similar system.

Keywords
Design patterns, pattern languages, interaction design, interdisciplinary design teams, guidelines, music, exhibits

INTRODUCTION: CHI AND COMMUNICATION

CHI deals with many aspects of communication. Not only is it trying to improve the communication between human and computer, but it also often stands at the borderline between software engineering and user during development.

Still, one of the major limits that CHI has not managed to break yet lies in communication. It is difficult for user interface experts to communicate their experience and methods to other design team members, from software engineering, to application domain experts. This leads to acceptance problems between these groups. They result from a lack of understanding of the paradigms, methods, and values of the other profession.

THE IDEA: A PATTERN APPROACH

Alexander originally introduced pattern languages into urban architecture [1] to create a vocabulary of proven and reusable design solutions for his profession. Software engineering picked up this idea, especially with object-oriented design patterns as presented by Gamma et al. [4]. HCI has only started to adopt Alexander's ideas, but initial efforts appear very promising. A more in-depth discussion of pattern languages is beyond the scope of this paper.

We believe, however, that the concept of creating a pattern language can and should be applied not only to architecture, software engineering, or HCI in isolation. Pattern languages are a more general model of structuring design knowledge. They can be used in any discipline that requires structured, creative work of some kind to be carried out. In particular, many interesting application domains for which software is being created, e.g., nontrivial office tasks, management work, or artistic activity, can be described as a pattern language. We can consider a composer who is writing music to be a "designer" of a musical artifact.

This observation leads us to our central idea: If those three distinct fields — application domain, HCI, and software engineering — each express their knowledge, experience, and values in the form of a *pattern language,* then the following improvements become possible (see Fig. 1):

1. Views and concepts of each profession become easier to understand for the others because the pattern concept of presentation is the same, and it is geared towards clarity.
2. Once the three pattern languages are understood, it will be easier to find bridges and analogies between them: Abstract concepts from the application domain may give hints to high-level user interface considerations, which in turn may indicate overall system characteristics. Concrete facts and objects from the application domain may point to concrete physical interaction details, which can be mapped to specific software objects and implementations.

The second step above joins the three distinct pattern languages together to a single structure that represents the combined experience of the interdisciplinary design team.

Naturally, these pattern languages do not (and should not) develop overnight. Rather, they will gradually evolve while successful solutions are being developed, usually over the course of several similar projects. The advantage of such a structured representation is that this knowledge is being captured to be used for a design rationale, to build a

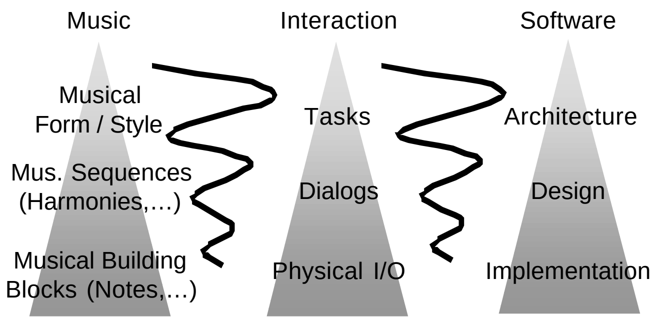

Fig. 1: A pattern language spanning three disciplines.

corporate memory, to train new designers in academic institutions or in industry, etc. It supplies an easy-to-remember vocabulary for communication about successful, reusable design solutions across discipline boundaries in a flexible form. For example, Alexander's original pattern collections about urban architecture are readable and fascinating even for non-architects.

PROOF-OF-CONCEPT: MUSIC EXHIBIT DESIGN

Music is an application domain not closely connected to software engineering or HCI. We chose this area to prove that the pattern language idea can be used to structure the knowledge in such a domain as well.

As part of our research, we have developed several interactive exhibits for public display. *WorldBeat,* an exhibit about making music with computers in new ways, was one of the first, and received the 1998 international *Multimedia Transfer Award*. The entire exhibit is controlled using a pair of infrared batons. Users can conduct music, play virtual drums and other instruments, hum a tune to find it in a database, and even improvise to a Blues band without playing wrong. More details can be found in [2] and the video proceedings of that CHI conference.

The system was built with some pattern ideas already in mind. These patterns dealt with musical concepts, and how these concepts could be represented in the interaction and in the internal software architecture [3].

However, we now look at our three disciplines from a unifying pattern perspective. We cannot explain all our patterns in detail here, but will go through the disciplines, each in a top-down fashion, following the hierarchy of a pattern language. What most patterns aim at will become clear from their name. Where applicable, the major components of a pattern will be mentioned: The *Context* in which the pattern can be applied, the *Forces* or conflicting goals that are to be met, the *Solution* to this problem, its *Consequences,* positive and negative, *Related patterns* that either use this pattern, that are used by it, or that can be used before or after it, and *Examples* of where this pattern has been observed or applied successfully.

Musical Design Patterns

In music, our application domain, the *Jazz Style* is an example of a high-level, abstract pattern. Its context is that a musical piece is to be created, with conflicting forces like ease-of-listening and musical complexity. The solution incorporates many lower-level patterns of jazz music for chord progressions (like the *II-V-I Progression*), instrumentation, rhythm, etc. These patterns reach down to low-level building blocks: individual notes (*Blue Note* pattern), rhythmic timings (*Triolic Groove* pattern), etc.

Interaction Design Patterns

An abstract pattern for interaction design is *InteractiveExhibit.* It captures the competing forces in the context of designing a public, computer-based exhibit: Visitor "throughput", interaction duration, information goals, and exterior constraints of the exhibition environment.

Its solution will involve the medium-level patterns *AttractUser, EngageUser,* and *DeliverMessage*, who in turn lead to patterns like *ExplorableInterface:* It balances the forces of a simple initial system impression (to implement *AttractUser*) and a system that remains interesting over longer periods of interaction time (*EngageUser),* by letting the user discover the system by himself. No sequential paths through the system are mandatory, initially only few options are there to choose from, and only in reaction to the user's actions further depth of the system is being revealed.

This gradual revealing finally can be implemented using the *DynamicDescriptor* pattern (examples are BallonHelp under MacOS, or ToolTips under MS Windows).

Software Design Patterns

This domain does not require much explanation as the idea of design patterns in software engineering is fairly established. In our case, these patterns would again create a hierarchical language, from high-level architectural patterns, to patterns about object interaction at a medium level of abstraction [4], down to coding patterns that deal with implementation aspects.

Linking the Pattern Languages

To create an interdisciplinary language from these three pattern languages, they need to be cross-linked. For example, the high-level concept of a certain musical form (e.g., arrangement in heads and choruses) will influence the overall design of the user's interaction with the system, and also its overall system architecture that will need to represent, store and process these concepts effectively. On the other hand, low-level concepts like a certain rhyhmic "groove" can be represented by a single user interface object (e.g., a slider), and handled by a set of basic time-processing modules within the software architecture.

CONCLUSIONS AND FURTHER RESEARCH

A pattern language spanning several disciplines promises to be a very useful method for capturing design experience in interdisciplinary teams. We have applied the idea to the design of interactive music exhibits with some first success.

To complete and verify our patterns, we are currently developing a new music exhibit with an entirely different theme (composing classical music) to see how well our patterns adapt to varying design situations.

We are also formalizing the interconnection between the different pattern languages, as a basis for computer-based development tools, but with human readability in mind.

REFERENCES

1. Alexander, C. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, UK, 1977.
2. Borchers, J. WorldBeat: Designing a Baton-Based Interface for an Interactive Music Exhibit. *Proc. CHI'97 (*Atlanta GA, March 1997), ACM Press, 131–138.
3. Borchers, J., and Muehlhaeuser, M.: Design Patterns for Interactive Music Systems. *IEEE Multimedia* ***5****(3),* IEEE Computer Society, 1998, 36–46.
4. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading MA, 1995.