Interdisciplinary Design Patterns

Jan O. Borchers

University of Linz
4040 Linz, Austria
+43 732 2468 744
jan@tk.uni-linz.ac.at
http://www.tk.uni-linz.ac.at/~jan

Abstract

Designing successful interactive systems requires user interface experts to work together with developers and users in an interdisciplinary team. However, these groups usually miss a common terminology to exchange ideas, opinions, and values. This paper proposes that each of these groups express its design experience and paradigms as a design pattern language, which makes it easier for the other disciplines to understand those issues.

Introduction

Pattern languages have proven to be a medium that is very well suited to the task of communicating design experience and knowledge even to outsiders of the respective profession. In fact, one of Alexander's original goals in publishing his pattern language for urban architecture was to allow the inhabitants (that is, the *users*) to participate in designing their environments [1].

Software engineering picked up the concept around 1987 [2], and that community has since had a large number of software design patterns collected and published, especially in the PLoP conference series [3]. The idea of using patterns to empower users to participate in design decisions, and the goal of creating systems and environments of a higher quality of use, however, has largely been lost, and software patterns become hardly more than a convenient format for discussion amongst designers. To quote Alexander's observations on pattern usage in software engineering:

> "Now, my understanding of what you are doing with patterns... It is kind of a neat format and that is fine. The pattern language that we began did have other features, and I don't know whether those have translated into your discipline. I mean there was at root behind the whole thing a continuous mode of preoccupation with under what circumstances is the environment good. In our field that means something." [4]

HCI has picked up the pattern idea earlier than many people would expect [5,6,7]. Recently, it has begun to receive broader attention [8,9], and HCI pattern languages have started to appear [10]. It turns out that user interface design is much closer to architecture than software design, because it deals more directly with spatial relationships and visual aesthetics. However, the importance of the time dimension distinguishes UI design quite fundamentally from architecture, and it is not clear yet how temporal relationships can be cast into patterns [7,8].

An interdisciplinary pattern approach

Pattern languages have thus shown their usefulness for expressing design experience, in a format also understandable for nonprofessionals, in architecture, software engineering, and user interface design. We propose that in every application domain that incorporates some form of creative, design-like work, expertise can be expressed in the form of patterns as well.

We therefore suggest that HCI professionals, software engineers and application domain experts or users should express their respective design experience, paradigms, and values as pattern languages. This format makes it easier to understand each other's design criteria and concepts, fostering interdisciplinary communication that is crucial for the design of a successful interactive system.

Example: Interactive Music Exhibit

We developed an interactive exhibit about music called *WorldBeat* [11], and used the pattern format to express not only software and UI design issues, but also the knowledge from the application domain "music" for this project [12,13]. In music, the "designer" is a composer or player creating musical artifacts, and in Jazz, for example, a language of patterns can describe important concepts, from large-scale ("twelve-bar blues") down to small-scale ("triplet groove") issues.

A formal, graph-based definition of patterns and pattern languages that should help focusing the discussion about patterns, and simplify the construction of computer tools to work with pattern languages, without impeding their readability by humans, is proposed in [13].

Once all three disciplines involved in the project - software engineering, user interface design, and application experts (musicians in this case) - have expressed their experience in pattern format, the human factors people can exchange their language with the application experts to arrive at a UI design that closely resembles concepts of the application domain, and they can exchange their language with software engineering to arrive at a system design that supports the intended features and interactivity in an ideal way. The actual project environment, such as "designing an interactive exhibit", is the concrete context in which the patterns are applied (Fig. 1).

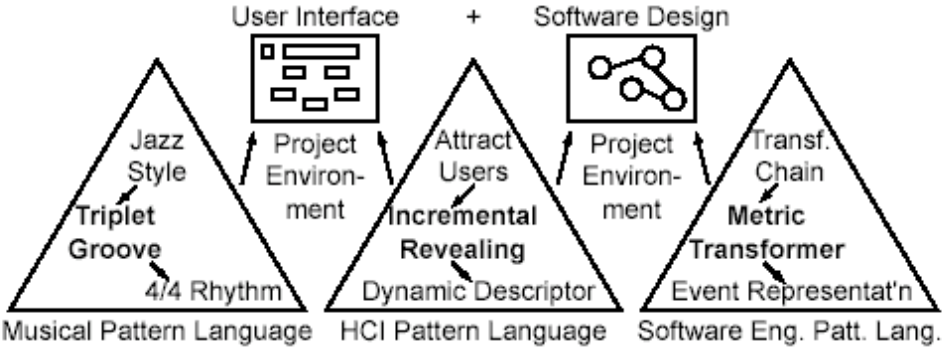


Fig. 1: Design pattern languages (selected patterns shown) for Music, HCI, and software engineering are used in a project environment "interactive exhibit" to create a user interface and software design.

Naturally, patterns cannot capture the intuition and creativity of an expert designer in any domain, but they can help important basic design principles. Also, these languages need to evolve over time. Nevertheless, in our followup projects, the pattern form not only simplified communication between the different disciplines involved, but also helped to introduce new members of the design team into the experience and findings from past projects, building a useful corporate memory.

Didactic pattern use

Two further advantages have become clear in using the pattern form in our work: First, HCI patterns have served well to communicate basic UI design knowledge to students. In a course about user interface design by the author, first-year computer science undergraduates received a pattern collection [10], and they were able to quickly relate many of those patterns to their own design projects and concrete problems. In a poll among the 32 students at the end of the course, the usefulness of HCI patterns for understanding UI design concepts, the ease of relating them to their own projects, and their suitability for use in future projects were each rated with an average "2" (1=very good...5=very bad).

Second, our WorldBeat exhibit used the pattern form to didactically structure its presentation of application domain (musical) concepts. These concepts were embodied into user interface objects and relationships that users can see and interact with. This supported active "learning by doing" and made the musical concepts easier to understand.

References

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

2. Beck, K. and Cunningham, W., *Using Pattern Languages for Object-Oriented Programs,* OOPSLA'87 workshop on Specification and Design for Object-Oriented Programming, 1987. <http://c2.com/doc/oopsla87.html>

3. Coplien, J.O. and Schmidt, D.C., *Pattern Languages of Program Design,* Software Patterns Series, Addison-Wesley, 1995 (and subsequent volumes).

4. Alexander, C., Keynote Address, *OOPSLA'96 Conference,* 1996. Video available from ACM.

5. Norman, D.A., *The Psychology Of Everyday Things,* Basic Books, New York, 1988, p. 229.

6. Apple Computer, *Macintosh Human Interface Guidelines,* Addison-Wesley, 1992, Appendix B. <ftp://ftp.apple.com/devworld/Technical_Documentation/Human_Interface/Human_Interface_Guidelines.sit.hqx>

7. Barfield, L., van Burgsteden, W., Lanfermeijer, R., Mulder, B., Ossewold, J., Rijken, D. and Wegner, P., Interaction design at the Utrecht School of the Arts, *SIGCHI Bulletin,* **26**(3), 1994, pp. 49-79.

8. Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., Gross, B., Lehder, D., Marmolin, H., Moore, B., Potts, C., Skousen, G., and Thomas, J., Putting it all together: Towards a pattern language for interaction design, *SIGCHI Bulletin,* **30**(1), 1998, pp. 17-23.

9. Borchers, J., CHI Meets PLoP: An Interaction Patterns Workshop (ChiliPLoP'99 Conference, Wickenburg, AZ), *SIGCHI Bulletin* (to appear), 1999. <http://www.tk.uni-linz.ac.at/~jan/patterns/chiliplop99-report-ps.zip>

10. Tidwell, J., Interaction Design Patterns, *PLoP'98 Conference on Pattern Languages of Programming* (Illinois), 1998. <http://www.mit.edu/~jtidwell/interaction_patterns.html>

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

12. Borchers, J. and Mühlhäuser, M., Design Patterns for Interactive Musical Systems, *IEEE MultiMedia* **5**(3), IEEE Computer Society, 1998, pp. 36-46.

13. Borchers, J., Designing Interactive Music Systems: A Pattern Approach, *Proc. HCI International '99* (Munich, Germany), 1999 (in print).

Sample Pattern: Incremental Revealing

... if you are designing an *Interactive Exhibit,* a computer-based public system for museums and similar institutions that is accessed by many people for a relatively short time and only once, then you try to *Attract Users* and *Engage Users,* especially by conveying an initial *Simple Impression.* Now, however, it is time to decide how to unfold the contents and features of your interactive system over the course of interaction to meet these design goals.

***

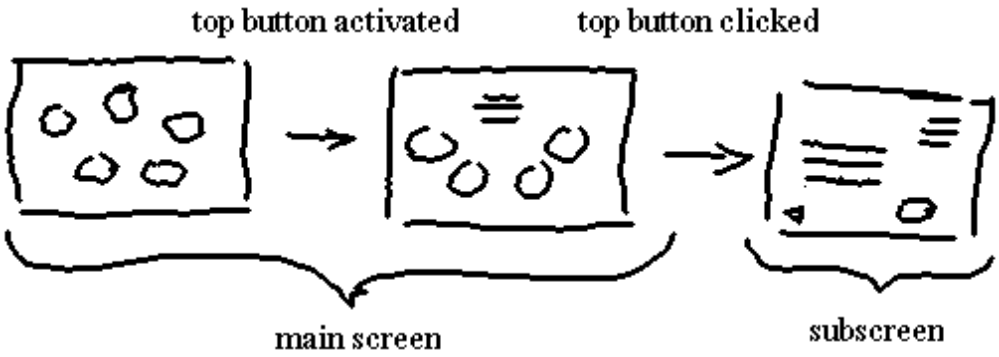
If a system looks difficult to use, people will not go near it in the first place. But if it appears too simple, even after the visitor has explored it a little, he will be bored and leave it behind quickly.

A simple impression is important to make a system look non-intimidating and inviting, especially for novices. Somebody who does not feel at home with operating computers or other devices will shy away from using a system that looks complex and only usable for experts. A multitude of initial options, features, and information will keep non-experts from even approaching your system, and you have no chance of even trying to deliver your message with your system to those people.

But after the initial contact, to keep your visitors engaged, your exhibit needs to convey its depth of features and contents as well. Once the visitor has started to explore the system and interact with it, it needs to show that it has more to offer, and that there are more rewarding experiences to be expected if the user interacts with the system for a while. If the system does not seem to have anything more to offer, then the user will think that it does not make sense to spend any more time with it. The message that you wanted to convey with your exhibit may not have been delivered yet.

These two forces contradict each other. To balance them, a system needs to gradually reveal its complexity to the user.

The WorldBeat exhibit has a very simple main selection screen from where the user starts to explore the exhibit. It only contains short names and icons for the various features of WorldBeat: composing, conducting, improvising, etc. Only when the visitor moves the cursor of his infrared baton over one of these items, a short phrase explains what to expect behind this button. Finally, only when the visitor actually clicks on this button, the system switches to the new page (subscreen) where the feature can be tried out and explored in detail.



WorldBeat revealing screens.

Even most desktop GUIs initially show a menu bar only. Nowadays, many applications can be used with hardly using the actual menu entries that are hidden behind that bar. Only if the user clicks onto a menu, the actual multitude of commands available becomes visible. Even more complex settings only appear in dialogues when the user issues commands that require those settings.

Therefore:

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.

***

An additional stage of Incremental Revealing, between the initial page and the subsequent page, can be easily inserted by using *Dynamic Descriptors* which show what lies behind a user interface object without the user having to actually use it (see the WorldBeat example, or MacOS Balloon Help). It will also be easier to implement Incremental Revealing when you arrange the contents of your interactive exhibit into a *Flat & Narrow Tree*

structure. Finally, make sure to provide an opportunity to get back from the more complex parts to the initial easy overview *(Closed Loop),* so the visitor knows that he has understood this part of the exhibit *(Deliver Message)*....

Author: Jan O. Borchers <jan@tk.uni-linz.ac.at>. Last modified 22.07.1999 16:41