CHI Meets PLoP: An Interaction Patterns Workshop

Jan O. Borchers

To appear in ACM SIGCHI Bulletin, Vol. 32, No. 1, January 2000

Abstract

This report summarizes the results of a workshop on pattern languages for human-computer interaction which took place at the ChiliPLoP'99 Conference on Pattern Languages of Programming. It suggests a definition and taxonomy for interaction patterns, explains how Writers' Workshops are used to improve patterns, and points out some surprising issues about pattern languages as they are understood by key players in that field. It shows the importance of user interface and software engineering researchers to exchange their thoughts on this hot topic.

What's ChiliPLoP?

The annual ChiliPLoP conference in Arizona is part of the *Pattern Languages of Programming (PLoP)* conference series, its partner conferences being the annual main PLoP conference in Chicago, and the EuroPLoP which is held in (surprise) Europe. ChiliPLoP'99 took place in Wickenburg, Arizona, from 16 to 19 March 1999. Common goal of these conferences is to gather and discuss design patterns from software engineering and related areas. Two techniques are used to accomplish this: First, experienced pattern writers *shepherd* the pattern drafts of new authors before the conference, especially for PLoP. Second, submitted patterns are discussed at the conference in *Writers' Workshops* in presence of the author.

This year, ChiliPLoP consisted of a Newcomers track, and a Hot Topics track. Newcomers were instructed about the basics of writing patterns by renowned pattern authors. The Hot Topics were workshops about specific pattern areas. They included Agent Patterns, Elementary Patterns (goal: a patterns book accompanying introductory computer science courses at university), Interaction Patterns, Component Design Patterns, Organizational Patterns, and Telecommunications Patterns (TelePLoP). Some of these Hot Topics, such as TelePLoP, have been continued for several years already, whereas others like Interaction Patterns were new to the conference.

The Interaction Patterns workshop consisted of six participants. I turned out to be the only "CHI person", with all others being rooted in software engineering with an interest in HCI issues. This made the meeting quite different from the first workshop at CHI'97 on this subject [3], and also made it a particular interesting experience for me, as it was up to me to explain and defend the paradigms of HCI...

Results of the Interaction Patterns Workshop

Definition: Interaction Pattern Language

To explain what pattern languages are about in general is beyond the scope of this report. Briefly, a design pattern captures a proven solution to a recurring design problem in a generative and easy-to-understand, human-readable format. A pattern language is a hierarchically structured collection of design patterns that leads the designer from abstract, large-scale to concrete and small-scale design issues. Design patterns have proven to be a very suitable medium (or literary form, really) to communicate design experience and the design values of their author, in architecture where the idea first emerged as well as in many areas of software engineering that have picked up this concept. The best way to learn about pattern languages is to have a look at architect Christopher Alexander's original books on the subject, which are a fascinating read even for non-architects [1],[2].

Our workshop started out by trying to define what it means to create pattern languages for human-computer interaction design. After much discussion, we arrived at the following definition, which also found broad approval by the other pattern people at the conference:

> An Interaction Pattern Language generates space/time interaction designs that create a system image close to the user's mental model of the task at hand, to make the human-computer interface as transparent as possible.

A Taxonomy of Interaction Patterns

We agreed upon three main dimensions along which interaction patterns can be classified meaningfully (see Fig. 1).

The most important dimension is *level of abstraction:* Interaction design patterns can address very large-scale issues that comprise a user's complete *task*, they can address smaller-scale, slightly more concrete topics that describe the *style* of a certain part of the interaction (such as the *Browser* style identified by workshop organizer Bill Brooks), or they

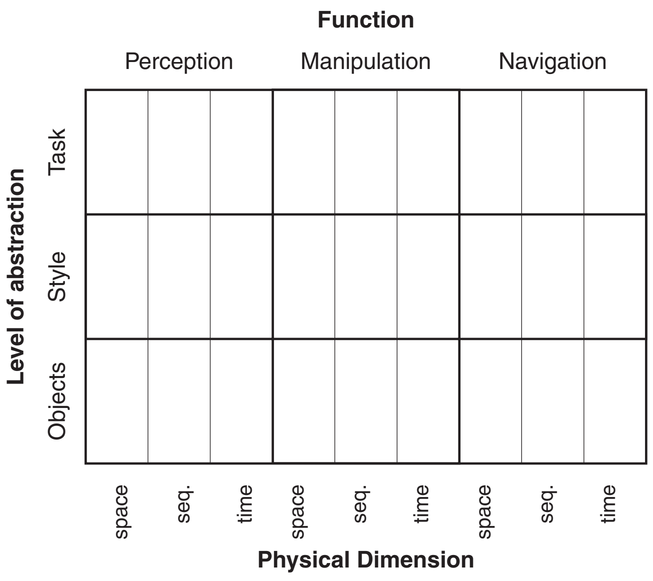

Figure 1: A taxonomy of human-computer interaction design patterns.

can deal with low-level questions of user interface design that look at individual user interface objects (whether virtual or physical).

We discussed the inclusion of a fourth layer here which would be called "technology", to distinguish the actual input and output hardware considerations, but finally decided against it since the distinction from software objects did not seem useful enough.

The second fundamental dimension is *function:* Patterns can be classified into those that address mainly questions of (visual, auditory, etc.) *perception* (interface output), and those that deal with interface input, or, more specifically, *manipulation* of some kind of application data, or *navigation* through the system.

Human Factors people will probably consider this, especially the disctinction between navigation and manipulation, a bit unusual and too software-centered, but it has to be kept in mind that the whole work was as a result of software engineering and CHI paradigms being brought together, and measured against each other!

The third dimension that we identified is *physical dimension:* Some patterns will address questions of *spatial* layout, while others deal with issues of *sequence* (discrete series of events, e.g., a sequence of dialogs), or with continuous *time* (such as a design pattern about good animation techniques in the user interface).

This taxonomy is the result of some iterations in which we looked at interaction patterns in the sense of our definition that some participants had submitted, and tried to sort them into our classification scheme.

For example, the pattern *Incremental Revealing* captures the idea that a user interface for non-expert users should initially appear relatively simple and easy to grasp, and that the system should only reveal additional "depth" (contents or features) when the user becomes active and looks for it. The pattern is described in more detail in [4].

This pattern lies at a *high level of abstraction:* it addresses how the complete *task* of the user is dealt with. Its *function* lies mainly in *perception* as it suggests how much information to display or otherwise output to the user. Its *physical dimension* is *sequence* as it deals with the distribution of this information over a sequence of user events, e.g., subsequent screens of an information system.

Writers' Workshop

This is a quite formal process widely used in the software patterns community to discuss the patterns submitted by an author: After an initial *welcoming,* the author first *reads* a part of her work to the authors, to remind everybody of the person behind the work. All critics, who usually are pattern authors themselves, have read the paper before the workshop. After this introduction, the author fades into the background and attends the following discussion without interfering (called a *fly on the wall*). One of the critics now *summarizes* the paper in his own words; the others can add to this summary. Next, *positive comments* on *form* and *content* of the submission (what should be kept), and subsequently constructive *negative* comments on form and content (what could be improved) are collected. The discussion ends with a summary of the good points of the paper (this *sandwich technique* avoids a negative lasting impression). After this, the author is *welcomed back* into the group and allowed to ask *questions* if some comments were not clear to her, or if she wishes to see another aspect of her paper discussed. She is not allowed, however, to defend her work at this time. As this whole discussion can be a bit harsh at times, the author is finally *applauded* for her work (and braveness to submit it), and somebody closes the session with an entertaining *unrelated story.*

We held Writers' Workshops on two of our paper drafts. For me, it turned out to be a very useful activity that gave me many constructive suggestions for improving my paper – as an author, you get the chance to observe how others actually interpret and understand your text, and where problems arise. The whole technique, by the way, is taken from the world of literature where it is used to discuss an author's poems; Richard Gabriel has carried it over to the area of software patterns.

Literature Review

The organizers handed out a CD with a collection of existing research papers and pattern collections on the subject. Most of them can be found at [5].

Other Activities

Two other talks at the conference were of particular importance for interaction design patterns.

Write Languages, Not Patterns

In one of his presentations, James Coplien pointed out that the connections between patterns are at least as important as the patterns themselves. An isolated pattern does not make much sense; only a language of patterns can capture the quality of a system as a whole. Therefore, a pattern author should not try just to write individual patterns, but, starting out from existing systems, try to extract their positive qualities and cast them into a hierarchical pattern language.

It is interesting to note that Coplien and numerous others at that conference did not regard the well-known design patterns book by the *Gang of Four* [6] as a pattern language; some went so far as to claim that it might not even contain patterns at all. The reason stated for this is that the book does not really describe the principles of object-oriented design, but rather a collection of workarounds to put into reality certain concepts of object oriented design using an incomplete OO language such as C++. Therefore, it is probably not a good idea to teach object-oriented design in courses by just going through this book.

Alexander on the Moral Aspects of Patterns

Another highlight was the videotaped talk from the OOPSLA'96 conference by the inventor of the pattern concept, architect Christopher Alexander, who put forward the following thesis: Even if all architects around the world were to pick up his language of patterns, he would not reach his original goal of improving the quality of life measurably, because architecture only influences a small part of people's environments. Computer science, on the other hand, touches on an increasingly large part of everyday life. Therefore, the computing community should not just use patterns as a "nice way of describing software design knowledge", but also pick up Alexander's original goal: To strive for systems with the *Quality Without A Name* that improve the quality of everyday life. This goal should be the starting point of every pattern language, and every software design. To me, it seems that the ethics discussion actually has a better chance of entering the software engineering world via this new vehicle of design patterns.

Summary and Future Work

It became clear that the software engineering patterns community requires more input from other disciplines such as HCI. Nevertheless, the atmosphere at the conference was very open and interdisciplinary already. The workshop participants plan to put existing material and findings together at a dedicated internet domain. A workshop on patterns in human-computer interaction at INTERACT'99 in Edinburgh in September, which the author co-organizes, will be another important step to bring the pattern idea to our discipline. The long-term goal is to have an online repository of documented, interlinked patterns and pattern languages for user interface designers to access for help in their daily work.

Acknowledgements

The findings of this workshop would not have been arrived at without the cooperation of all workshop participants, and I would like to thank the other participants for this: Bill Brook, who organized the workshop, Andrew Carlson, and Anthony Flaks (all of AT&T Labs, UK) Todd Coram (of Blacksmith,Inc., USA), and Jens Coldewey (of Coldewey Consulting, Germany).

References

[1] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[2] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[3] Elisabeth Bayle, Rachel Bellamy, George Casaday, Thomas Erickson, Sally Fincher, Beki Grinter, Ben Gross, Diane Lehder, Hans Marmolin, Brian Moore, Colin Potts, Grant Skousen, and John Thomas. Putting it all together: Towards a pattern language for interaction design. *SIGCHI Bulletin*, 30(1):17–23, 1998.

[4] Jan Borchers. Designing interactive music systems: A pattern approach. In *Proceedings of the HCII'99 8th International Conference on Human-Computer Interaction*, Munich, Germany, August 22–27 1999.

[5] Thomas Erickson. Interaction patterns home page. Established February 1998. http://www.pliant.org/ personal/Tom_Erickson/InteractionPatterns.html.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

About the Author

Jan Borchers is a researcher at the Telecooperation Group at the University of Linz in Austria. He works at user interface design issues for new media, and currently develops an interdisciplinary pattern-based approach to designing interactive systems that will help software engineers, user interface designers, and application domain experts to talk to each other. He has also managed several projects in desigining computer-based interactive exhibits, such as the award-winning *WorldBeat* music exhibit at the Ars Electronica Center in Linz, and authored a number of publications on these issues. Currently, he works as visiting scientist and lecturer for Human-Computer Interaction courses at the University of Ulm in Germany.

Author's Address

Jan O. Borchers
Telecooperation Group
University of Linz
Altenberger Str. 69
4040 Linz, Austria
Phone: +43 732 2468 744
Fax: +43 732 2468 9829

Visiting Scientist at:
Distributed Systems Department
University of Ulm
James-Franck-Ring
89069 Ulm, Germany
Phone: +49 731 502 4192
Fax: +49 731 502 4142

Email: jan@tk.uni-linz.ac.at
WWW: http://www.tk.uni-linz.ac.at/∼jan/