A Pattern Approach to Interaction Design

Jan O. Borchers
Department of Computer Science
Darmstadt University of Technology
Alexanderstr. 6, 64283 Darmstadt, Germany
jan@informatik.tu-darmstadt.de

ABSTRACT

To create successful interactive systems, user interface designers need to cooperate with developers and application domain experts in an interdisciplinary team. These groups, however, usually miss a common terminology to exchange ideas, opinions, and values.

This paper presents an approach that uses pattern languages to capture this knowledge in software development, HCI, and the application domain. A formal, domain-independent definition of design patterns allows for computer support without sacrificing readability, and pattern use is integrated into the usability engineering life cycle.

As an example, experience from building an award-winning interactive music exhibit was turned into a pattern language, which was then used to inform follow-up projects and support HCI education.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces—*style guides; theory and methods; training, help, and documentation; user-centered design*; H.5.5 [**Information Interfaces and Presentation (e.g., HCI)**]: Sound and Music Computing—*modeling, systems*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*user interfaces*

General Terms

Design, Human Factors, Documentation

Keywords

Pattern languages, design methodologies, interdisciplinary design, music, exhibits, education

1. INTRODUCTION

To design systems that fulfil today's high expectations concerning usability, human-computer interaction (HCI) experts need to work together very closely with team members from other disciplines. Most notably, they need to cooperate with application domain experts to identify the concepts, tasks, and terminology of the product environment, and with the development team to make sure the internal system design supports the interaction techniques required.

However, these disciplines lack a common language: It is difficult for the user interface designer to explain his guidelines and concerns to the other groups. It is often even more problematic to extract application domain concepts from the user representative in a usable form. And it is hard for HCI people, and for application domain experts even more so, to understand the architectural and technological constraints and rules that guide the systems engineer in her design process. In general, people within a discipline often have trouble communicating what they know to outsiders, but to work together well, disciplines must learn to appreciate each other's language, tradition, and values [23].

1.1 Pattern Languages as *lingua franca*

Simply stated, a pattern is a proven solution to a recurring design problem. It pays special attention to the context in which it is applicable, to the competing "forces" it needs to balance, and to the positive and negative consequences of its application. It references higher-level patterns describing the context in which it can be applied, and lower-level patterns that could be used after the current one to further refine the solution. This hierarchy structures a comprehensive collection of patterns into a *pattern language*.

The central idea presented here is that HCI, software engineering, and application domain experts in a project team each express their expertise in the form of a pattern language. This makes their knowledge and assumptions more explicit, and easier for the other disciplines to understand and refer to. Such a common vocabulary can greatly improve communication within the team, and also serve as a corporate memory of design expertise.

The next section briefly explains the concept and history of pattern languages. A critical component of these languages, however, are the cross-references that help readers find their way through the material. To facilitate creating and navigating through patterns, possibly with computer support, a formal syntactic definition of patterns and their relations is presented that is independent of the domain they address. It is also shown where in the usability engineering life cycle

DIS '00, Brooklyn, New York.

patterns can be applied.

The final section shows a pattern language for interactive music exhibits as an example, and summarizes some initial empirical studies about the usefulness of pattern languages in HCI courses.

2. A BRIEF HISTORY OF PATTERN LANGUAGES

This section briefly outlines where the pattern idea comes from, and how it has been adapted to other disciplines.

2.1 Patterns in Urban Architecture

The original concept of pattern languages was conceived by architect Christopher Alexander in the 1970's. In [2], he explains how a hierarchical collection of architectural design patterns can be identified to make future buildings and urban environments more usable and pleasing for their inhabitants. In [3], he presents 253 such patterns of "user-friendly" solutions to recurring problems in urban architecture. They range from large-scale issues ("COMMUNITY OF 7000", "IDENTIFIABLE NEIGHBOURHOOD"), via smaller-scale patterns ("PROMENADE", "STREET CAFE") down to patterns for the design of single buildings ("CASCADE OF ROOFS", "INTIMACY GRADIENT"). Finally, in [4], the architect uses his pattern language to define a new planning process for the University of Oregon as an example.

It is less known that Alexander's goal in publishing this pattern language was to allow not architects, but the inhabitants (that is, the *users*) themselves to design their environments. This is strikingly similar to the ideas of user-centered and participatory design, which aim to involve end users in all stages of the software development cycle.

Pattern languages essentially aim to provide laymen with a vocabulary to express their ideas and designs, and to discuss them with professionals. This idea of creating a vocabulary implements well-known results from psychological research about *verbal recoding*: "When there is a story or an argument or an idea that we want to remember [...], we make a verbal description of the event and then remember our verbalization" [24]. The idea can be recalled when its short name is remembered.

Each of Alexander's patterns is presented as several pages of illustrated text, using a very uniform structure and layout with the following components [3, p. x]:

A meaningful, concise *name* identifies the pattern, a *ranking* indicates the validity of the pattern, a *picture* gives a "sensitizing" and easily understood example of the pattern applied, and the *context* explains which larger patterns it helps to implement. Next, a short *problem statement* summarizes the competing "forces", or design tradeoffs, and a more extensive *problem description* gives empirical background information, and shows existing solutions.

The following *solution* is the central pattern component. It generalizes the examples into a clear, but generic set of instructions that can be applied in varying situations. A *diagram* describes this solution and its constituents graphically, and *references* point the reader to smaller patterns that can be used to implement this pattern.

It must be stressed that Alexandrian patterns are, above all, a didactic medium for human readers, even (and especially) for non-architects. This quality must not be lost in a more formal representation or extension of the idea to other domains.

2.2 Patterns in Software Engineering

Around 1987, software engineering picked up the pattern idea. At the OOPSLA conference on object-orientation, Beck et al. [8] reported on an experiment where application domain experts without prior Smalltalk experience successfully designed their own Smalltalk user interfaces after they had been introduced to basic Smalltalk UI concepts using a pattern language. It is interesting to note that this first software pattern experiment actually dealt with user interface design, and user participation.

The workshop started a vivid exchange about software design patterns. An influential collection of patterns for object-oriented software design was published by Gamma et al. [18]. The annual Pattern Languages of Programming (PLoP) conferences have established an entirely new forum to exchange proven, generalized solutions to recurring software design problems.

The overall format of a pattern has not changed very much from Alexander to, e.g., Gamma et al. [18]: Name, context, problem, solution, examples, diagrams, and cross-references are still the essential constituents of each pattern.

The goals, however, have changed in an important way, without many noticing: Software design patterns are considered a useful language for communication *among software developers*, and a practical vehicle for introducing less experienced developers into the field. The idea of end users designing their own (software) architectures, has not been taken over. On the one hand, this makes sense, because people do not live as intensely "in" their software applications as they live in their environments. On the other hand, though, a good chance to push the concept of participatory design forward by introducing patterns has not been taken advantage of. This was one of the reasons why, at OOPSLA'99, the authors of that book were put before a mock "trial" for their work. (See the article by Tidwell [31] for an interesting comment.)

2.3 Patterns in HCI

The pattern idea has been referenced by HCI research earlier than most people expect. Norman and Draper [28] mention Alexander's work, and in his classic *Psychology of Everyday Things* [27, p. 229], Norman states that he was influenced particularly by it. Apple's Human Interface Guidelines [5] quote Alexander's books as seminal in the field of environmental design, and the Utrecht School of Arts uses patterns as a basis for their interaction design curriculum [6].

But only recently a first workshop dedicated to pattern languages for interaction design took place within the HCI community. It showed that the ideas about adopting the pattern concept for HCI were still very varied, and that "as yet,

there has been little attention given to pattern languages for interaction design" [7]. The patterns reported by this workshop were necessarily not strictly design patterns, but rather activity patterns describing observed behaviour (at the conference), without judging whether these represented "good" or "bad" solutions. Similarly to the Utrecht curriculum designers [6], it identified the temporal dimension as making interaction design quite different from architectural design. The workshop also stressed the often underestimated fact that patterns, to a large extent, represent the *values* of their author, i.e., the qualities that the author considers important in the artefacts he designs.

Subsequent workshops at UPA'99 [19], INTERACT'99 [11], and at CHI 2000 [14] have confirmed the growing interest in pattern languages within the HCI community, and helped to more precisely shape the notion of HCI design patterns and their use. A preliminary definition of HCI design patterns, and suggestions for structuring HCI design pattern languages, as well as a sample format of an HCI pattern, were given at the INTERACT workshop. The CHI 2000 workshop refined these findings and definitions, and the pattern idea has also been linked to other related concepts of usability engineering, such as *claims* [29].

Meanwhile, a number of concrete pattern collections for interaction design have been suggested. The language by Tidwell [30], for example, covers a substantial field of user interface design issues. Interaction design patterns have found their way into the PLoP conference series, becoming even a "hot topic" at the ChiliPLoP'99 conference [10].

Less research, though, has gone into formalizing pattern languages for HCI, to make them more accessible to computer support. Even Alexander [3, p. xviii] admits that, "since the language is in truth a network, there is no one sequence which perfectly captures it". The hypertext model of a pattern language presented here makes it possible to create tools for navigating through the language, similar to those for interface design guidelines [1, 21].

2.4 Patterns in the Application Domain

Patterns have been a successful tool to model design experience in architecture, in software design (with the limitations discussed here), and, as existing collections such as [30] show, also in HCI. Other domains have been addressed by patterns as well. Denning and Dargan [16], in their theory of action-centred design, suggest a technique called *Pattern Mapping* as a basis for cross-disciplinary software design. Referring to Alexander's work, they claim that patterns could constitute a design language for communication between software engineers and users, just as Alexander's pattern language does between builder and inhabitant. Granlund and Lafrenière [20] use patterns to describe business domains, processes, and tasks to aid early system definition and conceptual design. The authors also note the interdisciplinary value of patterns as a communications medium, and the ability of patterns to capture knowledge from previous projects.

Indeed, there is no reason why the experience, methods, and values of any application domain cannot be expressed in pattern form, as long as activity in that application domain includes some form of design, creative, or problem-solving work. This brought the author to the idea that not only HCI professionals and software engineers, but also application domain experts, could express their respective experience, guidelines, and values in the form of pattern languages.

The history of pattern languages in architecture, software engineering, and especially HCI is described in more detail in [12].

3. USING PATTERN LANGUAGES IN INTERDISCIPLINARY DESIGN

The following sections give a general model of pattern languages, and show how to integrate those pattern languages into the usability engineering process.

3.1 A Formal Hypertext Model of a Pattern Language

A formal description of patterns makes it less ambiguous for the parties involved to decide what a pattern is supposed to look like, in terms of structure and content. It also makes it possible to design computer-based tools that help authors in writing, and readers in understanding patterns.

We first give a formal syntactic definition:

- A *pattern language* is a directed acyclic graph (DAG) $\mathbf{PL}=(\mathbf{P},\mathbf{R})$ with nodes $\mathbf{P} = \{P_1, \ldots, P_n\}$ and edges $\mathbf{R} = \{R_1, \ldots, R_m\}$.
- Each node $P \in \mathbf{P}$ is called a *pattern*.
- For $P, Q \in \mathbf{P}$: P *references* $Q \iff \exists R = (P, Q) \in \mathbf{R}$.
- The set of edges leaving a node $P \in \mathbf{P}$ is called its *references*. The set of edges entering it is called its *context*.
- Each node $P \in \mathbf{P}$ is itself a set $P = \{n, r, i, p, f_1 \ldots f_i, e_1 \ldots e_j, s, d, \}$ of a name n, ranking r, illustration i, problem p with forces $f_1 \ldots f_i$, examples $e_1 \ldots e_j$, the solution s, and diagram d.

This syntactic definition is augmented with the following semantics:

Each **pattern** of a language captures a recurring design problem, and suggests a proven solution to it. The language consists of a set of such patterns for a specific design domain, such as urban architecture.

Each pattern has a **context** represented by edges pointing to it from higher-level patterns. They sketch the design situations in which it can be used. Similarly, its **references** show what lower-level patterns can be applied after it has been used. This relationship creates a *hierarchy* within the pattern language. It leads the designer from patterns addressing large-scale design issues, to patterns about small design details, and helps him locate related patterns quickly.

The **name** of a pattern helps to refer to its central idea quickly, and build a vocabulary for communication within a

team or design community. The **ranking** shows how universally valid the pattern author believes this pattern is. It helps readers to distinuish early pattern ideas from truly timeless patterns that have been confirmed on countless occasions.

The opening **illustration** gives readers a quick idea of a typical example situation for the pattern, even if they are not professionals. Media choice depends on the domain of the language: Architecture can be represented by photos of buildings and locations; HCI may prefer screen shots, video sequences of an interaction, audio recordings for a voice-controlled menu, etc.

The **problem** states what the major issue is that the pattern addresses. The **forces** further elaborate the problem statement. They are aspects of the design that need to be optimised. They usually come in pairs contradicting each other.

The **examples** section is the largest of each pattern. It shows existing situations in which the problem at hand can be (or has been) encountered, and how it has been solved in those situations.

The **solution** generalizes from the examples a proven way to balance the forces at hand optimally for the given design context. It is not simply prescriptive, but generic so that it can generate a solution when it is applied to concrete problem situations of the form specified by the context.

The **diagram** supports the solution by summarizing its main idea in a graphical way, omitting any unnecessary details. For experts, the diagram is quicker to grasp than the opening illustration. Media choice again depends on the domain: a graphical sketch for architecture, pseudo-code or UML diagram for software engineering, a storyboard sketch for HCI, a score fragment for music, etc.

With these definitions, a formal model for pattern languages is in place. However, formalization must not impede readability and clarity of the material. The process of writing patterns should not be hindered by the formal notation, and the results should still be accessible in a variety of formats, including linear, printed documentation (which most people still prefer for sustained reading for well-known reasons). Each part of a pattern, and its connections to other patterns, are usually presented as several paragraphs in the pattern description (see, for example, [3]). Other media, such as images, animations, audio recordings, etc., are used to augment the pattern description as described above.

3.2 Using Patterns in the Usability Engineering Lifecycle

It is not necessary to follow a specific design method to use this pattern language approach. As Dix et al. [17, p. 6] point out, "... probably 90% of the value of any interface design technique is that it forces the designer to remember that someone (and in particular someone else) will use the system under construction." Nevertheless, the design process must emphasize usability, and therefore include programmers and UI designers as well as end users [32, p. 57]. Also, as with most usability methods [17, p. 179], patterns will not be used in just a single design phase, but throughout the entire design process.

A suitable process framework for this is Nielsen's *usability engineering lifecycle* model [26, p. 72]. Patterns fit into each of the eleven activities that this model suggests:

1. Know the user. If the application domain of an interactive software project involves some designing, creative, or problem solving activity, then its concepts and methods can be represented as a pattern language in the sense of the above definition. The development team, after explaining the basic idea of patterns to user representatives, can begin eliciting application domain concepts in the form of patterns from those experts. Those patterns do not need to be perfect in terms of their timeless quality. They just give a uniform format to what needs to be captured anyway, but explicitly stating problems, forces, existing solutions, and references within these "work patterns". Also, patterns for user interface and software design will have the same format, making all the material more accessible to all members of the design team, and helping users to recognize their work patterns in user interface concepts of the end product.

2. Competitive analysis. Here, existing products are examined to gather information and hints for the design of the new system. The internal architecture of competing products is usually not accessible, but user interface design solutions of successful competing systems can be generalized into HCI design patterns for the new product.

3. Setting usability goals. The various aspects of usability, such as memorability, and efficiency of use, need to be prioritized. They can, however, be used as forces of abstract HCI design patterns which explain how these forces conflict, and how this conflict is to be solved for the current project. A design for a system used intensively by expert users, for example, will put the balance between the above goals more towards efficiency of use.

4. Parallel design. Several groups of designers can develop alternative user interface prototypes to broaden the "design space" explored. HCI design patterns can serve as a common ground, working as design guidelines to ensure that the usability goals from the last phase are fulfilled. These patterns can also come from external sources such as HCI design books.

5. Participatory Design. This technique involves user representatives, or application domain experts, in the design process to evaluate prototypes, and participate in design discussions. These users will have few problems understanding the pattern language of their application domain (which they have probably helped to create themselves). But knowing this format will also help them understand the HCI patterns that the user interface design team has collected, and which represents their design values, methods, and guidelines. Conversely, the UI design team can use the application domain pattern language to talk among themselves and to users about issues of the application domain, in a language that users will find resembles their own terminology. A common vocabulary for users and user interface designers emerges from the combination of both languages.

6. Coordinated design of the total interface. Coordinated design aims to create a consistent interface, documentation, online help and tutorials, both within the current and with previous versions of this and other products within a product family. HCI design patterns, especially those addressing lower-level, concrete design guidelines, can serve as vocabulary among design teams, to help ensuring this consistency. Of course, additional methods such as dictionaries of user interface terms are required to support this process.

7. Apply guidelines and heuristic analysis. Style guides and standards are the ways to express HCI design experience that are closest to HCI design patterns. Patterns can improve these forms through their structured format and contents, combination of concrete existing examples and a general solution, and an insightful explanation not only of the solution, but also of the problem context in which this solution can be used, as well as the structured way in which individual patterns are integrated into the hierarchical network of a pattern language. This coverage of multiple layers of abstraction and expertise is similar to the distinction between general guidelines, category-specific rules which are derived from previous projects, and product-specific guidelines that are developed as part of an individual project [26, p. 93].

8. Prototyping. Prototyping puts concrete interfaces into the hands of users much earlier than the final product, albeit limited in features, performance, and stability. In this more software-oriented area of usability engineering, software design patterns play an important role. If the development group members express their architectural standards and components, as well as specific project ideas, as patterns, then the user interface design group can relate to those concepts more easily, and will better understand the concerns of the development team. For example, the HCI design group could change the feature set in a prototype to make it easier to implement without compromising its usefulness for testing.

9. Empirical testing. Prototypes, from initial paper mockups to the final system, are tested with potential users to discover design problems. While patterns cannot help the actual evaluation process, the set of HCI patterns can be used to relate problems discovered to the patterns that could be applied to solve those problems, as shown in the next phase.

10. Iterative design. Prototypes are redesigned and improved iteratively and based upon feedback from user tests. In this activity, patterns of HCI or software design experience are an important tool to inform the designer about his options. Contrary to general design guidelines, which are mainly *descriptive*, and merely state desirable general features of a "good" finished interactive system, patterns are *constructive*: they suggest how a problem can be solved. Naturally, patterns evolve over the course of a project, reflecting the progress in understanding of the problem space and improving the design. Successful solutions serve as examples for existing patterns, or initiate the formulation of a new pattern. Subsequent projects will relate more easily to patterns with such well-known examples. The result is a post-hoc "structural" design rationale which keeps the lessons learned during a project accessible for the future. Patterns are less suited to document good and bad design decisions in the form of a "process" design rationale, although the concept of *anti-patterns* of bad, but tempting solutions could be used to model discarded design options.

11. Collect feedback from field use. Studies of the finished product in use, help line call analysis, and other methods can be used to evaluate the final product after delivery. Here, the application domain pattern language serves as an important tool to talk to users. HCI design patterns point out design alternatives for solutions that need to be improved. Similarly, feedback can strengthen the argument of those patterns that created a successful solution, and suggest to rethink those that led to suboptimal results.

4. EXAMPLE: DESIGNING INTERACTIVE MUSIC EXHIBITS

As a proof of concept, we used our experience from designing an interactive, computer-based music exhibit, to start building a pattern language about the musical, HCI, and software design lessons learned from that project. We will briefly present the original project's goals and its design, and then give some examples of resulting patterns from the various disciplines, and at various levels of abstraction.

4.1 The WorldBeat Project

The *Ars Electronica Center* in Linz, Austria [22] is a technology museum "of the future", an exhibition and venue centre where the arts and new technology merge. Our research group designed one floor within this centre, addressing future computer-supported learning and working environments. Apart from an electronic class/conference room [25], we designed several exhibits showing the use of computers in specific learning subjects and working situations. *WorldBeat* was one of these exhibits, designed by the author.

Briefly, *WorldBeat* allows the user to interact with music in quite new ways. The complete exhibit is controlled using just a pair of infrared batons. They are used to navigate through the various pages of the system, and to create musical input, from playing virtual instruments like drums or a guitar, to conducting a computer orchestra playing a classical piece, to improvising to a computer Blues band – without playing wrong notes. Furthermore, users can try to recognize instruments by their sound alone, and locate tunes by humming their melody. The system is described in more detail in [9], and on the video proceedings of that CHI conference. The goal of *WorldBeat* was to show visitors that computers may open up entirely new ways of interacting with musical information, many of which they can expect to see implemented in future consumer products.

To design a system for such an environment, we had to take into account a number of factors that do not usually have to be considered in such detail when designing an interactive system. However, instead of listing our findings as a loose set of guidelines, we used the pattern format described above to summarize our experience in all three disciplines. We will show what musical concepts we identified to address in our exhibit, describe our user interface patterns specific to

this scenario, and present software design solutions that we found for this system.

We will not describe the patterns in full detail; that would typically require several pages per pattern. Instead, we show the network of patterns identified so far, and look in more detail at the main issues of context, problem, solution, examples, and references, for a small but representative selection. The actual patterns are written in a more detailed textual form without explicit labels for "Context", "Problem" etc.: Instead, they use implicit typographical structuring to clearly show the components of each pattern. The complete pattern languages can be found in [12].

4.2 Musical Design Patterns

We will begin by describing the musical concepts and "guidelines" that musicians use when they compose or improvise music. Our point is that this process can be considered a "design" activity as well, and that it is feasible to structure the rules and values of this design process into patterns.

At the most abstract level we consider useful, a certain style of music is chosen. Our exhibit features various musical styles in different parts of the system, but we will restrict ourselves to the presentation of the BLUES STYLE. The patterns below are referenced by that top-level style pattern. Downward links are "references" relations (see Fig. 1).

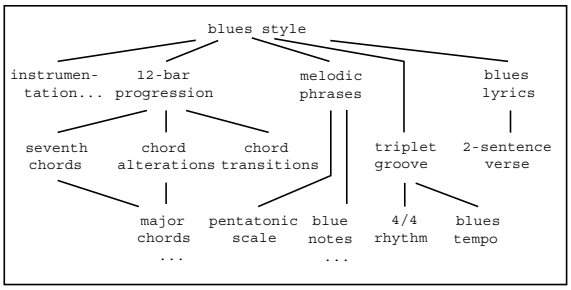

Figure 1: A pattern language for blues music.

TWELVE-BAR PROGRESSION

Context: Playing a BLUES STYLE piece.

Problem: Players need to agree on a common sequence ("progression") of musical chords to create harmonically coherent music. A progression is useful to avoid a too static, boring impression, but it should be simple enough to be easily remembered while playing.

Solution: Use the following Blues progression of chords, each lasting one bar (in C major, I = C major, IV = F major, V = G major):

I-I-I-I-IV-IV-I-I-V-V-I-I

Examples: The above is the simplest form of any Blues piece, and most Blues music adheres to it. It is found in countless recordings and sheet music.

References: The sequence is built from basic MAJOR CHORDS. Many variations of this basic progression are possible that make the music more interesting. A simple one is to use SEVENTH CHORDS instead of the simple major chords. The sequence can be varied further by replacing chords with more complex CHORD TRANSITIONS.

TRIPLET GROOVE

Context: Playing a piece in the BLUES STYLE. The concept is also used in other styles like Swing, and Jazz in general.

Problem: Players need to create a swinging rhythmic feeling. The straight rhythm from other styles does not create this. At the same time, sheet music cannot include all rhythmic variances because it would become too complex and unreadable.

Solution: Where the written music contains an evenly spaced pattern of eighth notes, shift every second eighth note in the pattern backwards in time by about one third of its length, shortening it accordingly, and make the preceding eighth note one third longer. Instead of a rhythmic length ratio of 1:1, the resulting pattern are alternating notes with a length ratio of $\frac{2}{3} : \frac{1}{3}$. Two straight eighth notes have been changed into 2+1 "triplet" eighth notes. This rhythmic shift creates what musicians call the "laid-back groove" in a performed piece. Fig. 2 shows this concept in traditional notation.

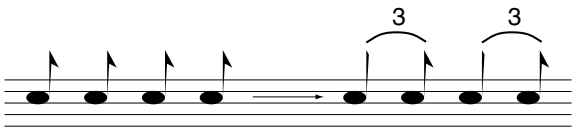

Figure 2: From straight notation to triplet groove.

Examples: Any recorded Blues piece will feature this rhythmic shift, although the actual shift percentage varies very widely. Usually, the faster a piece is, the less shifting takes place.

References: TRIPLET GROOVE always modifies an underlying straight beat, typically a 4/4 RHYTHM.

These examples show what issues musical patterns may address, and how they can be formulated. We will now look at HCI design patterns that could create an interface dealing with such musical concepts.

4.3 Interaction Design Patterns

This section describes our collection of patterns for HCI design. We have focused on issues particularly important for interactive exhibits, but they are of equal importance to "kiosk" and similar public-access systems where typical users are first-time and one-time users with short interaction times and no time for a learning curve.

The outer graph (see Fig. 3) shows most of them, with top-down links again representing "references" relations. We will go into more detail for two of these patterns.

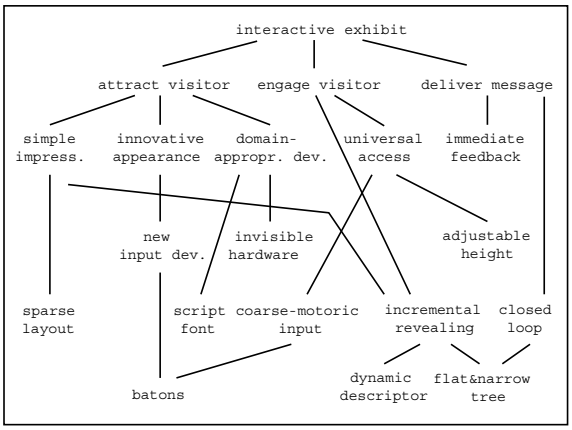

Figure 3: An HCI pattern language for interactive exhibits.

ATTRACT VISITOR

Context: An INTERACTIVE EXHIBIT that should attract and engage its visitors to deliver a message.

Problem: In an exhibition centre, many exhibits are "competing" for the visitor's attention although they should rather "cooperate" to inform the visitor, without one system becoming too dominating in its appearance. On the other hand, people will never discover the message of an exhibit if they are not drawn towards it in the first place, or if it looks too complicated. After all, there is usually nothing that forces the visitor to use any of the exhibits.

Solution: Make the exhibit look interesting by creating an innovative-looking interface that promises an unusual experience, but make it appear simple enough to scare off neither computer nor application domain novices. Use an appearance and interaction technique that is adequate for the domain of the exhibit (which is usually not Computing).

Examples: At the *WorldBeat* exhibit, there is no mouse, keyboard, or computer visible; all that the user sees is a pair of infrared batons, and a monitor with a simple, inviting startup screen (see Fig. 4).

Figure 4: A user at the WorldBeat exhibit.

References: The SIMPLE IMPRESSION pattern shows how to build a system that does not scare off visitors. INCREMENTAL REVEALING conveys initial simplicity without limiting the depth of the system. Use DOMAIN-APPROPRIATE DEVICES so the exhibit and its periphery reflect its subject area.

INCREMENTAL REVEALING

Context: Designing a computer exhibit interface that ATTRACTS VISITORS with its initial SIMPLE IMPRESSION, but that still ENGAGES VISITORS for a while with sufficient depth of functionality and content.

Problem: A simple appearance, and presenting the system's depth are competing goals.

Solution: Initially, present only a very concise and simple overview of the system functionality. Only when the user becomes active, showing that he is interested in a certain part of this overview, offer additional information about it, and show what is lying "behind" this introductory presentation.

Examples: *WorldBeat* is structured into a short introductory screen, followed by a simple main selection screen with only names and icons of the main exhibit components (conducting, improvising, etc.) If the user moves the pointer towards one of the component icons, a short explanation appears (first revealing stage). Then, if he selects it, a separate page opens up that explains the features in more detail (second revealing stage).

References: Most information systems reveal their content incrementally through a FLAT AND NARROW TREE structure. The DYNAMIC DESCRIPTOR pattern also implements incremental revealing. It can be found in Apple's Balloon Help, Windows ToolTips, and has also been identified in [30].

4.4 Software Design Patterns

As Gamma et al. [18] suggested, domain-specific software design patterns are important to supplement the general ones. There are many general software patterns that could be identified in our system, but we will concentrate here on those patterns that relate specifically to software design for music exhibits. We will describe one of these patterns below. More details on these patterns can be found in [15] and [12].

METRIC TRANSFORMER

Context: Musical performance adds many subtle variations to the lifeless representation of a written score, in the harmonic, melodic, and rhythmic dimensions. To model these variations, a system will follow the TRANSFORMER CHAIN pattern, where sequence of transformations are applied to an incoming stream of musical data. Due to its one-dimensional nature, the rhythmic dimension is especially accessible for computer modeling.

Problem: An incoming stream of MUSICAL EVENTS needs to be modified in its timing: Some events need to be delayed for a short period. The delay may follow a deterministic algorithm, or a random distribution.

Solution: A number of objects need to interact for such

a functionality: A Creator supplies the raw, straight musical material, and a Metronome the raw, straight rhythm. A Modulator models the metric transformation, i.e., the deviation from the uniform beat. It captures the musical idea behind the transformation. It may feed overall gradual tempo changes back to the Metronome. A Customizer lets the user change the Modulator parameters in real time. A Timer takes the basic Metronome beat, and modifies it with the Modulator output. A Player component finally outputs the rhythmically transformed musical material as a new sequence of MUSICAL EVENTS.

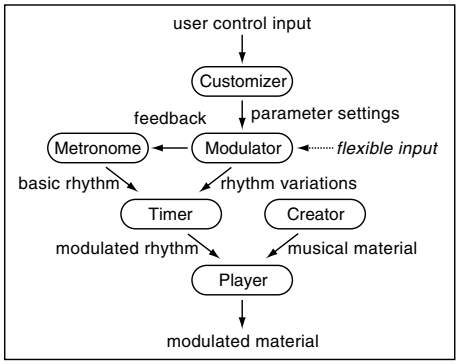

Figure 5: The metric transformer pattern.

Examples: The *WorldBeat* exhibit features a *Groove* slider that lets the user change the swing feeling of a Blues band while playing (see below). Other examples of such variations that the pattern could implement are the random variations of human performance when interpreting a written score, or the conducting process where the conductor makes subtle changes to the existing tempo and dynamic structure of an orchestra piece.

References: This pattern works best with a representation of the musical material as MUSICAL EVENTS; an example is the widely used Musical Instruments Digital Interface (MIDI) standard.

4.5 Reusing the Pattern Language

In a follow-up project, we designed an interactive music exhibit to learn about the concept of the classical *fugue*. The musical domain is quite different from jazz, and only a few musical patterns carried over into the new project. However, we identified many HCI and software engineering patterns in our language that carried over very well to this new task. In particular, these patterns helped us communicate our previous experience to new members of the design team. Recently, we finished work on two other projects in a similar domain for the HOUSE OF MUSIC VIENNA, including a system to conduct the Vienna Philharmonic Orchestra [13], and have been able to again reuse and refine many of the patterns that were identified through *WorldBeat* and improved in the Fugue project.

4.6 Pattern Use in Education

HCI patterns were also used by the author to teach user interface design to first-year computer science undergraduate students. Although these students only received a pattern collection to refer to for their first design projects, it was surprising to see how quickly they discovered patterns that were applicable to their current design situation, how they used those proven solutions in their own project context, and how a vocabulary of patterns was used quickly to discuss options and solutions within and between the design groups. In an ad-hoc quiz and poll at the end of the term, students were able to recall 1.6 patterns by name on the average (although patterns had only been used for a short period of the course), and judged their usefulness with an overall 1.9 (1 = will absolutely reuse them, 5 = will absolutely not reuse them). The findings of this study are described in more detail in [12].

5. CONCLUSIONS AND FURTHER RESEARCH

Communication in interdisciplinary design teams is a major problem for HCI practitioners. We suggested that all team members – especially HCI people, software engineers, and experts from the application domain – formulate their experience, methods, and values in the form of pattern languages, as originally introduced in urban architecture. For a uniform representation and computer support, we proposed a formal representation of patterns and their relations.

As an example, we created pattern languages from our experience in creating an interactive music exhibit. Our results are promising, and indicate that such patterns help people from outside the respective discipline to understand our findings. They have proven useful to create subsequent exhibits with a similar background, and HCI patterns have served well in an HCI course.

Currently, we are extending and refining our language by identifying additional patterns and relations. First steps have also been undertaken in designing the Pattern Editing Tool, PET, a system to support creating, reviewing, and browsing pattern languages.

A complete version of the HCI and other pattern languages for interactive exhibits and similar systems, as well as a description of PET, can be found in [12].

Applying the pattern technique to entirely new application domains could further strengthen our argument that structuring them into patterns is a generally valid approach, and a more intensive use of HCI patterns in user interface design courses will hopefully give us more detailed findings about their usefulness in education.

6. ACKNOWLEDGEMENTS

The author would like to thank his colleagues at the Distributed Systems Group at the Darmstadt University of Technology, the Telecooperation Research Group at the University of Linz, and at the Distributed Systems Group at the University of Ulm, particularly Max Mühlhäuser at Darmstadt, for their help in working on the projects that created the necessary basis of this work.

Portions of this article have been adapted from Work to be published in full in "A Pattern Approach to Interaction Design" by the author, to be published by John Wiley & Sons, October 2000.

7. REFERENCES

[1] L. Alben, J. Faris, and H. Sadler. Making it Macintosh: Designing the message when the message is design. *Interactions*, 1(1):10–20, January 1994.

[2] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[3] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[4] C. Alexander, M. Silverstein, S. Angel, S. Ishikawa, and D. Abrams. *The Oregon Experiment*. Oxford University Press, 1988.

[5] Apple Computer. *Macintosh Human Interface Guidelines*. Addison-Wesley, 1992.

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

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

[8] K. Beck and W. Cunningham. Using pattern languages for object-oriented programs. Technical Report CR-87-43, Tektronix, Inc., September 17, 1987. Presented at the OOPSLA'87 workshop on Specification and Design for Object-Oriented Programming.

[9] J. O. Borchers. WorldBeat: Designing a baton-based interface for an interactive music exhibit. In *Proceedings of the CHI 97 Conference on Human Factors in Computing Systems (Atlanta, GA, USA, March 22–27, 1997)*, pages 131–138, ACM, 1997.

[10] J. O. Borchers. CHI meets PLoP: An interaction patterns workshop. *SIGCHI Bulletin*, 32(1):9–12, January 2000. (Workshop at ChiliPLoP'99).

[11] J. O. Borchers. Interaction design patterns: Twelve theses, 2000. Position paper for the workshop *Pattern Languages for Interaction Design: Building Momentum*, CHI 2000 (The Hague, Netherlands, April 2–3,2000).

[12] J. O. Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons, October 2000.

[13] J. O. Borchers. The Virtual Conductor: Conduct the Vienna Philharmonic, 2000. http://go.to/virtualconductor.

[14] J. O. Borchers, R. N. Griffiths, L. Pemberton, and A. Stork. Pattern languages for interaction design: Building momentum. Workshop at CHI 2000 (The Hague, Netherlands, April 2–3,2000); report to be published, 2000.

[15] J. O. Borchers and M. Mühlhäuser. Design patterns for interactive musical systems. *IEEE Multimedia*, 5(3):36–46, 1998.

[16] P. Denning and P. Dargan. Action-centered design. In T. Winograd, editor, *Bringing Design to Software*, chapter 6, pages 105–119. Addison-Wesley, 1996.

[17] A. J. Dix, J. E. Finlay, G. D. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice-Hall Europe, London, second edition, 1998.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[19] Å. Granlund and D. Lafrenière. A pattern-supported approach to the user interface design process. Workshop report, UPA'99 Usability Professionals' Association Conf. (Scottsdale, AZ, June 29–July 2, 1999), 1999.

[20] Å. Granlund and D. Lafrenière. PSA: A pattern-supported approach to the user interface design process. Position paper for the UPA'99 Usability Professionals' Association Conf. (Scottsdale, AZ, June 29–July 2, 1999), 1999.

[21] R. Iannella. Hypersam: A practical user interface guidelines management system. In *Proc. QCHI'94 Second Annual CHISIG (Queensland) Symposium*, Bond Univ., Australia, 1994.

[22] S. Janko, H. Leopoldseder, and G. Stocker. *Ars Electronica Center: Museum of the Future*. Ars Electronica Center, Linz, Austria, 1996.

[23] S. Kim. Interdisciplinary cooperation. In B. Laurel, editor, *The Art of Human-Computer Interface Design*, pages 31–44. Addison-Wesley, 1990.

[24] G. A. Miller. The magical number Seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956. http://www.well.com/user/smalin/miller.html.

[25] M. Mühlhäuser, J. Borchers, C. Falkowski, and K. Manske. The Conference/Classroom of the Future: An interdisciplinary approach. In *Proc. IFIP Conf. "The Internat. Office of the Future: Design Options and Solution Strategies" (Tucson AZ, Apr. 9–11, 1996)*, pages 233–250. Chapman & Hall, 1996.

[26] J. Nielsen. *Usability Engineering*. Morgan Kaufmann, San Francisco, 1993.

[27] D. A. Norman. *The Psychology of Everyday Things*. Basic Books, New York, 1988.

[28] D. A. Norman and S. W. Draper. *User-Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[29] A. Sutcliffe and M. Dimitrova. Patterns, claims and multimedia. In *Human-Computer Interaction – INTERACT '99*, pages 329–335, Edinburgh, UK, 30th August–3rd September 1999. IOS Press, Amsterdam.

[30] J. Tidwell. Interaction design patterns. PLoP'98 Conference on Pattern Languages of Programming, Illinois, extended version at http://www.mit.edu/-~jtidwell/interaction_patterns.html, 1998.

[31] J. Tidwell. The Gang of Four Are Guilty. http://www.mit.edu/~jtidwell/gof_are_guilty.html, 1999.

[32] B. Tognazzini. *TOG on Interface*. Addison-Wesley, 1992.