To appear in *Proceedings of HCI International 2001 9th International Conference on Human-Computer Interaction,* August 5-10, 2001, New Orleans, USA

A Pattern-Supported Approach to the User Interface Design Process

Åsa Granlund
Ericsson Radio Systems AB
Box 1248
S-581 12 LINKÖPING, Sweden
asa.granlund@era.ericsson.se

Daniel Lafrenière
GESPRO Technologies
1245 chemin Sainte-Foy, Édifice 1
Québec (QC), Canada G1S 4P2
lafrenid@gespro.com

David A. Carr
Institutionen för Systemteknik
Luleå University of Technology
S-971 87 LULEÅ, Sweden
david@sm.luth.se

ABSTRACT

Patterns describe generic solutions to common problems in context. Originating from the world of architecture, patterns have been used mostly in object-oriented programming and data analysis. The goal of HCI patterns is to create an inventory of solutions to help designers (and usability engineers) to resolve UI development problems that are common, difficult and frequently encountered. In this paper, we present our pattern-supported approach to user interface design in the context of information visualization. Using a concrete example from the telecommunications domain, we will focus on a task/subtask pattern to illustrate how knowledge about a task and an appropriate interaction design solution can be captured and communicated.

1 INTRODUCTION

1.1 What Is a Pattern?

A pattern is a formalized description of a proven concept that expresses non-trivial solutions to a UI design problem. The primary goal of patterns in general is to create an inventory of solutions to help UI designers resolve UI development problems that are common, difficult and frequently encountered. (adapted from Loureiro & Plummer, 1999)

A pattern is a format for describing a solution to a design problem. Patterns originate from architecture and were introduced by Christopher Alexander (Alexander, et. al., 1977; Alexander, 1979) in the mid-70's. Alexander noticed that certain solutions always apply to the same recurring problems and developed patterns as a design knowledge documentation method.

Software Engineering (Gamma, Helm, Johnson & Vlissides, 1995) adopted the pattern as a way to facilitate reuse of software. Early attempts at reuse of components often failed because the units were too small and did not mesh well together. Software patterns were adopted to allow sharing of larger units, and they specify in quite fine detail how components interact. As such they are much more prescriptive than patterns for architecture.

User interface designers also noticed that certain design problems occurred over and over. These problems generally have known good solutions. However, there has been a problem communicating them. Guidelines represent a possible solution, but they are generally seen as hard to interpret and as requiring excessive effort to find relevant material (Mahemoff & Johnston, 1998). For this reason, there has been an increasing interest in patterns to document user-interface design solutions. The SIGCHI'97 workshop on patterns (Bayle, et. al., 1998) saw patterns as a way of dealing with the increasing complexity and diversity of HCI design.

Tidwell (1999) describes patterns as "… possible good solutions to a common design problem within a certain context, by describing the invariant qualities of all those solutions." Simply put, patterns can be said to provide powerful and generic design guidance in a format that is consistent and easy to read and understand – they convey knowledge about good design.

Patterns are used implicitly by many skilled UI designers who have found solutions that have worked for them in the past. However, these designers usually keep little in the way of formal (documented) descriptions of these solutions. Thus, there are in fact both *implicit* patterns and formal (*explicit*) patterns. Explicit patterns can be used as a means of collecting and *formalizing* this knowledge. Using HCI patterns for capturing and documenting design knowledge is currently a hot topic, and there are many reasons for this interest (Erickson, 1998):

- Patterns provide a *lingua franca* that can be read and understood by all, regardless of background.
- The existing formal ways of documenting UI design knowledge are often weak – patterns offer a good way of capturing and transferring this knowledge. They are presented consistently, are easy to read, and provide back-

ground reasoning. The format provides information about the problem at hand, the context, a solution and also the rationale behind this solution.

- They promote reuse.
- Patterns are a valuable source of information, supporting both the analysis and the current situation and the design of the new system.

However, we believe that patterns **cannot** serve as a single source of design knowledge. They must still be complemented by traditional sources of information. But, they will point to information that is generally valid (for a specific domain) and also to designs that have proven good for similar projects. Since the description format provides reasoning and motivation, a pattern's relevance for the current project can be tested and evaluated.

Patterns must also be part of a language of interrelated patterns, participating in and supporting each other, in order to be truly useful. The pattern language works on different scales, and promotes the iterative growth of a design.

1.2 HCI Patterns Are Different

Patterns differ from design guidelines. Guidelines aim at coherence among user interfaces by documenting all the intricacies of a particular user interface (such as Windows or MacOS). However, guidelines focus mostly on window/widget issues while neglecting the knowledge required for proper UI design. In fact, the major forces influencing design: the user, the context, and the task, are missing from guidelines. Design rationale is missing, too. Patterns capture and document all of this important knowledge. They are also more invariant over time.

HCI patterns also differ from other pattern types. Software engineering patterns tend to specify a very strong interrelationship among component descriptions. The emphasis is on interface specification among components. However, user-interface designers see these patterns as too rigid and too detailed. User interface designers are concerned with esthetics and social aspects as well as function. They also want freedom to innovate and express themselves. These desires, combined with the fact that HCI is a young discipline where much is unknown, make a pure engineering approach inappropriate. Thus, HCI patterns are closer to architectural patterns. However, HCI patterns are also tied to software systems and as such must take software issues into consideration.

1.3 The PSA Approach

Up to this point, most of the work on patterns in HCI has focused on screen design issues. Our pattern-supported approach (PSA) to the user interface design process suggests a wider scope for the use of patterns by looking at the overall user-oriented interface design process. PSA addresses patterns not only at the design phase, but *before* design. (See Figure 1.)

Based on the fact that the usability of a system emerges as the product of the user, the task and the context of use, PSA integrates this knowledge in most of its patterns, dividing the forces in the pattern description correspondingly (i.e., describing Task, User, and Context forces). PSA provides a *double-linked chain* of patterns (parts of an emerging pattern language) that *support* each step of the design process (Figure 2). For example, *task* patterns point to S*tructure and Navigation Patterns*, which in turn point to *GUI* D*esign Patterns*, and vice-versa. These patterns offer a way to capture and communicate knowledge from previous designs (including the knowledge from system definition, task/user analysis and structure & navigation design). Given a mature language of patterns belonging to the described classes, the PSA approach provides an entry point to this pattern language, and suggests (without restricting the pattern usage) a chain of appropriate patterns at different levels of analysis and design.

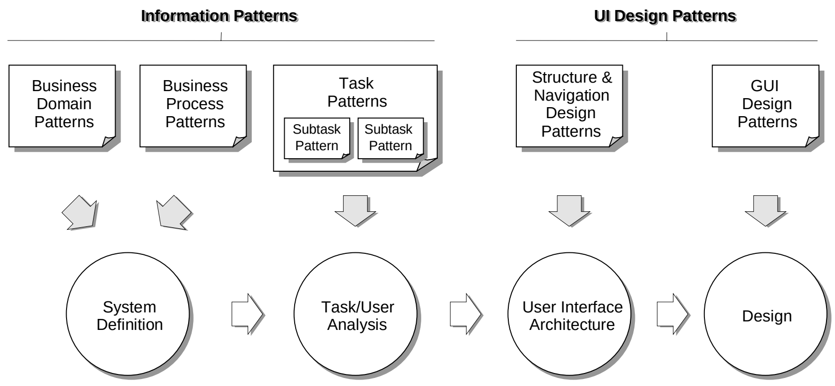

Figure 1 – The PSA Framework

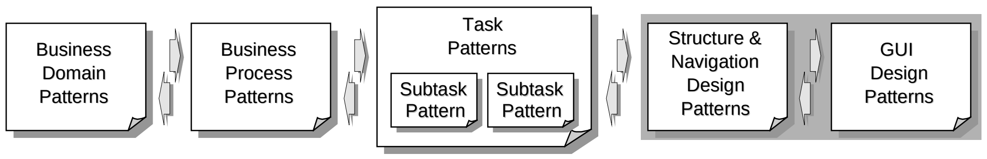

Figure 2 – Links between PSA Patterns

Here is a brief overview of the PSA patterns:

- **Business Domain Patterns** describe the type of business, its goals, plus the typical actors and business processes involved. They provide a starting point for initially defining the system design by pointing to relevant *Business Process* patterns and thereby to *Task* patterns. They help communicate the *System Vision*.
- **Business Process Patterns** describe typical processes and actors involved in the delivery of services/goods in compliance with the business goals. They narrow down the system definition and point to specific *Task* patterns to be considered.
- **Task Patterns** are used for capturing and passing on knowledge about the task, typical users, and their work context from previous similar projects, and for suggesting an appropriate interaction design solution. They point to *Structure & Navigation Design* patterns that describe solutions that have proven suitable for the task type in previous designs.
- **Structure & Navigation Design Patterns** describe ways to structure information and implement navigation in order to support the user's task. This design is based on the information described in the *Task* patterns.
- **GUI Design Patterns** document GUI design issues based upon information described in the *Task Patterns* and *Structure & Navigation Design Patterns*. They are based on the work of Tidwell (1999).

PSA is concerned with domain specific pattern support, that is: *Business Domain, Business Process* and *Task* patterns that describe a **specific** domain. We believe that this level of specificity is necessary – if the patterns are too abstract, they will not be useful in practice. However, it is desirable to make them as general and invariant as possible. Knowledge gained from this work will later be used to attempt the generalization that will also support transfer of design knowledge between domains.

Structure & Navigation Design patterns and *GUI Design* patterns, however, are **general** from the start, since they are not dependent on the domain. This means that the information in the *Business Domain* and *Business Process* and *Task* patterns leads up to suggestions for suitable *Structure & Navigation Design* patterns. *Structure & Navigation Design* patterns in turn point to *GUI Design* patterns, but the *Structure & Navigation Design* pattern in itself is **not** based on domain specific knowledge about task, user, and context.

2 FORMAT AND USE OF PSA PATTERNS

In order to demonstrate the use of PSA patterns, we will discuss the form of PSA patterns and illustrate their use with excerpts from a task pattern and a subtask pattern taken from our project to build a pattern language for system design for radio network management (mobile telephony). The patterns selected document part of the activity around optimizing the operation of the radio network. They also incorporate recommendations for the use of information visualization in this activity. The patterns incorporate guidance on design for visualization (Carr, 1999) into the context of managing cellular telephone networks.

We present only task and subtask patterns. In a design project, a set of relevant task patterns would have been selected during discussions with stakeholders, using Business Domain and Business Process patterns to guide the selection. The Task pattern can then be used for planning task and user analysis (and in some cases substitute for these activities, where for some reason, they cannot be carried out).

2.1 Task Patterns

Task patterns describe complex tasks, and point to subtask patterns that in turn describe parts of the task. They are used for capturing knowledge from previous similar projects, and use Description, Context and Forces to pass on the knowledge regarding: the task, typical users, their work context, and an appropriate interaction design solution. The Task pattern description consists of the following sections:

- **Name –** describes the task. A pattern called "Radio Network Optimization " would be a more specific instantiation of the more generic pattern "Network Optimization", which can apply to many different domains.
- **Context –** describes the goal of the design, specifying the user and the requirements of the task.
- **Problem –** describes the design problem at hand. Whenever there are competing concerns (forces) there will also be a problem – without the competing concerns, the solution is trivial, and there is no problem.

- **Example –** is used to clarify the task. This adds a feeling for the task at hand, and PSA tries to use a storyboard in order to make the example more vivid and easier to understand.
- **Forces –** describe all the, often conflicting, factors that influence design, directly or indirectly. PSA uses detailed forces, specific to the domain, but generic within it. The richness provides all the recurring information that would typically be gathered during task and user analysis. Information that is "project specific" does not fit into the Force description. Forces are divided into task, user, and context forces. The task forces describe task characteristics and special requirements. They build on general knowledge (e.g., regarding visualization tasks), but are made specific to the task (e.g., radio network optimization) in order to help the reader more clearly understand the problem. The reader avoids mental translation between general task characteristics and radio network optimization. The Force descriptions about the user are inspired by "personas" (Cooper, 1999), in that they paint a picture of a typical user. The information could easily be used as a basis for a full "persona" description in order to exemplify the user. Context forces are used to capture environmental and social factors that influence the work.
- **Design Solution –** communicates design considerations for the task and "guidelines" emerging from the forces. Since this pattern describes a complex task containing many subtasks, this interaction design solution (Figure 3) concerns overall interaction issues. For design on lower levels, each contained subtask pattern will contribute its own guidance.
- **Resulting Subtask Patterns –** describe the smaller tasks that are part of the complex task described in the pattern. These are generic, making up building blocks, while at the same time having their own pattern descriptions with forces, related Subtask patterns, and related Structure & Navigation patterns. For example, the Subtask patterns for our Optimize Radio Network pattern are: Graphic Overview, Zoom, Filter Details-on-demand, Relate, and History. They are derived from Shneiderman's visual information seeking tasks (Shneiderman, 1996).
- **Resulting Structure & Navigation Patterns –** suggest a way of navigating (and thereby implicitly structuring) the data/functions on the level of the complex task described in the pattern, in other words, the overall structure and navigation. Structure and navigation for parts of the task may well be different, and this is described in the resulting subtask patterns.
- **Resulting GUI Design Patterns –** suggest suitable patterns for GUI design solutions.

2.2 Subtask Patterns

All complex tasks are made up of smaller tasks – subtasks. Trying to describe a complex task in one pattern would be very difficult, and the description would quickly become large and unwieldy. Using the strength of a pattern language, we can let the task pattern point to subtask patterns that participate in the complex overall task. The subtask patterns are typically (at least from our experience so far) generic and not dependant on user task and context. This is due to the granularity of the description. They have the same components as the task pattern, describe solu-

The task forces emphasize the lack of structured workflow, the need to handle large amounts of data, and the need for displaying data based on the context of the specific network optimization problem. These forces and the specifics of the problem statement suggest information visualization as a solution. Information visualization is appropriate where the user has a task that is not easy to specify and involves large amounts of data (Carr, 1999). One can assume that the user will follow Shneiderman's information seeking mantra (Shneiderman, 1996): "Overview first, zoom and filter, then details on demand". This suggests the following organization for the design:

- Use a two-dimensional map image for overview. This map image should be as simple and uncluttered as possible, only providing the critical information at any given time.
- Provide zooming and panning of the map. Semantic zooming should be supported, where greater detail is revealed as the user zooms in.
- Provide flexible, direct-manipulation-based filtering. Dynamic queries (Ahlberg & Shneiderman, 1994) are an example design.
- Provide details-on-demand for all relevant objects in the map image. Make sure that the user doesn't lose spatial context when drilling-down.
- When using multiple views of the same data, make sure that their contents are synchronized. Consider a design similar to "snap-together visualizations" (North & Shneiderman, 2000) where users can quickly construct their own customized visualization consisting of multiple coordinated views.

Figure 3 – Design solution for the Optimize Radio Network pattern.

tions for interaction design and point to structure and navigation patterns at this lower level of detail. This should be quite intuitive – any complex system may have an overall structure, navigation model, and interaction design solution, and at the same time have subparts with varying solutions within it. The subtask pattern also refers to related subtask patterns.

There are two main differences between task and subtask pattern descriptions. Subtask forces are generally a subset of task forces or derived from them. They are specific to the subtask, and are not divided into Task, User, and Context Forces. At this level of detail the description is general for many complex tasks, regardless of the domain. From this point of view, forces are more general than for task patterns, and independent of task, user, or context. Also, subtask patterns have been found to be independent of a specific problem. Their design solution needs to be described in general terms

3 CONCLUSIONS AND FUTURE WORK

Up to this point, we have had only positive feedback from interaction designers presented with the approach. People appreciate the strength of the format, and believe it would really support them in their work. However, we have just started building a language of patterns, and many questions remain unsolved.

From a constructional point of view, we are currently working with the format of the structure and navigation patterns. Originally, the approach offered conceptual design patterns, but as these turned out to be too abstract to be useful, and we turned to the more practical subtask patterns. We are, however, striving to capture the more complex aspects of modeling.

We are also concerned with the robustness of the chain of patterns that we offer. What happens if some but not all forces apply? Can the link to the next level of patterns be trusted? We are thinking of having a "template" component for which pointers are always valid. But at the same time, we are concerned that the approach and related patterns will become too unwieldy. Our goal is that the user of the patterns should never be concerned with the construction of the patterns – they must be easy and intuitive.

But above all, the approach and the patterns need to be adapted and validated through practical usage. Today, the patterns do not fully supply a lingua franca, but are more or less targeting interaction designers. The descriptions, structure and level of detail must be adapted to fit actual design projects. This can only be done through iteration-based, practical use. In addition, whether or not the patterns can hold their promise of facilitating communication must be evaluated.

4 REFERENCES

Ahlberg, C. & Shneiderman, B. (1994) Visual information seeking: tight coupling of dynamic query filters with starfield displays, *Human Factors in Computing Systems Proceedings of CHI'94*, Boston, MA, 365-371.

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

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

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, *SGICHI Bulletin*, 30(1), 17-23.

Carr, D. (1999) Guidelines for designing information visualization applications, *Proceedings of ECUE'99*, Stockholm, Sweden.

Cooper, A. (1999). *The Inmates are Running the Asylum*. SAMS, Indianapolis, IN, ISBN 0-62-31649-8.

Erickson, T. (1998) Interaction Pattern Languages: A Lingua Franca for Interaction Design?, *UPA 98 Conference*, Washington, DC.

Gamma, E., Helm, R., Johnson, R., & Vlissides, R. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, ISBN 0-201-63361-2.

Loureiro, K. & Plummer, D. (1999). *AD Patterns: Beyond Objects and Components*. Research Note # COM-08-0111, Gartner Group.

Mahemoff, M, & Johnston, L. (1998) Principles for a usability-oriented pattern language, *OZCHI '98 Proceedings*, Adelaide, Australia, 132-139.

North, C., & Shneiderman, B. (2000) Snap-together visualization: a user interface for coordinating visualizations via relational schemata, *Proceedings of Advanced Visual Interfaces 2000,* Palermo, Italy, 128-135.

Shneiderman, B (1996). The eyes have it: a task by data type taxonomy for information visualizations, *Proceedings of 1996 IEEE Visual Languages,* Boulder, CO, 336-343.

Tidwell, J. (1999). *Common Ground: A Pattern Language for Human-Computer Interface Design*. http://www.mit.edu/~jtidwell/interaction_patterns.html