Seeking for structure in a Groupware Pattern Language

Till Schuemmer
FernUniversitaet Hagen
Computer Science VI - Distributed Systems
Universitaetsstrasse 1
55084 Hagen, Germany
+49-2331-987-4371
till.schuemmer@fernuni-hagen.de

ABSTRACT

Our collection of groupware patterns is currently growing to a large number of patterns. This paper explains our considerations for adding structure to this pattern language. It compares different approaches for structuring pattern languages and derives requirements for an authoring- and reading environment. The pattern navigator is such an environment, which is shown in a prototypical state.

INTRODUCTION

Our current activities in the area of groupware patterns have shown that the problem domain of groupware development includes a very large number of patterns. One reason for this may be the interdisciplinary nature of CSCW. The field brings together expertise from computer scientists, psychologists, sociologists, or organizational research to name only a few of them. A pattern language that aims to cover the whole groupware domain must therefore integrate all expertise that addresses the way how collaboration can be supported by computer technology.

During the search for groupware patterns, we found about 100 potential pattern candidates that we collected in a pattern map at [13] (note that the patterns map shown at the web site only shows one part of the current language). Discussions at pattern workshops showed that this map is not at least exhaustive. On the other hand, we already notice that the use of such a big pattern language is problematic. One problem that we observed was the problem of orientation – a well know issue in any non-linear piece of literature.

Readers of the pattern catalogue were not able to decide, which pattern they should read first, how the different patterns relate to each other, or which patterns are on what level of abstraction.

The self-containment of a pattern, which is one of its strengths when the pattern is applied, complicates the composition of different (isolated) patterns, if the users don't know, which patterns they should combine.

Initially, pattern languages were thought as a tool for lay people so that they could behave like a domain expert [3]. Future residents should be empowered to build their own houses that fit their needs. This learning process requires that the expert knowledge is presented in a way that does not confuse the reader or lets him get lost in the large quantity of self-contained knowledge bits.

We think that this problem is immanent to most larger pattern languages. Thus, we are interested in exploring different strategies for structuring pattern languages – especially with respect to the relations between patterns and the flow of reading of a user of the pattern language.

This position paper first provides an overview on different strategies for structuring pattern languages. The overview forms the basis for drawing requirements for a pattern authoring and reading environment in the second part of this paper. We finally present the pattern navigator, a prototype of such an environment.

CURRENT APPROACHES FOR STRUCTURING PATTERN LANGUAGES

The review of literature shows four different strategies for organizing a pattern language:

1. a linear sequence classified by chapters or pattern families,
2. a hierarchical structure of patterns,
3. a network of patterns that serves as a map, or
4. sequences of patterns.

The linear structure can be found in most books presenting a pattern language. For instance, the original publication of Alexander et al. [1] presents 253 patterns, which are ordered in three different areas: towns, buildings, and construction. The different areas group the patterns with respect to the size of space where they are applied – from towns as macroscopic spaces to small parts of buildings, such as windows, which are discussed in the construction section.

The linear structure was also used by Gamma et al. [8]. Here, design patterns are classified as creational, structural, and behavioural patterns.

The problem with using classification schemes for organizing pattern languages is that patterns can only be part of one class. This can result in a large number of classes that does not provide an additional reduction of complexity. In [6] for instance, the authors used a classification scheme with 10 classes containing 1 to 3 patterns. If the number of classes is kept small, it can lead to inconsistencies, as Tichy pointed out in [15].

Larger catalogues such as [15] are often structured using a hierarchical classification scheme. Patterns are first classified regarding the problem domain. Then, they are structured using various classification schemes – which ever meets the specific problem domain best.

Another application of hierarchical structures was proposed by Zimmer [17]: He introduces a uses-relation between patterns that reflects the case when one pattern uses another pattern in its solution. If one pattern uses another pattern, it is placed on a higher layer in the hierarchy. The resulting structure eases the understanding by decomposing the solution of high-level patterns using low level patterns. Note that – as in software decomposition – low level patterns are often used by more than one pattern in its solution.

Besides the uses relation, Zimmer defines two more classes of relations: the *Variant of X uses Y in its solution* relation and the *X is similar to Y* relation. He applies all three classes of relations to the Design-Patterns Catalogue of Gamma et al. [8] and generates a network of patterns. Although this network reveals dependencies between the problems and the solutions of patterns, it is not exhaustive and does not reflect all dependencies of the pattern language.

Another kind of network to the same set of patterns can be found in [8]. The authors generate this map by parsing the related patterns section of each pattern and inserting an edge between the patterns that is labelled with a short version of the explanation that was provided in the related patterns section.

Recent discussions in the area of architecture patterns focus on another means for guiding the user through a pattern language: the concept of sequences. Sequences are guided tours through a set of patterns, which provide the necessary glue to relate by definition isolated patterns. They are discussed in depth in [2].[1]

A possible way to implement sequences is the use of an example that runs all the way through the pattern language. One can find examples or case studies in many recent pattern languages, such as [12], [11], [16], [5]. Several pattern languages use a more abstract form of sequences. They don't use a concrete example but tell the reader, which patterns should be applied in a sequence (e.g. [7]). All cited examples of sequences use a textual form to describe the sequence.

The TimeTravel pattern language [4] interweaves a pattern sequence with the pattern descriptions themselves. The authors use icons to identify parts of the case study (the description of the problem domain) and meta-comments that describe the user who applies the pattern (as test cases and implementation parts). The patterns are embedded within these stories and the pattern description is reduced to a problem solution pair.

Although sequences serve as a good means for relating patterns in a concrete example, they only provide *one* example. The danger is that this example is taken literally and other relations between patterns are diminished in value.

Additional help for providing an overview of the pattern language is often given by adding all patterns' intent to a so-called intent catalogue. In [8] for instance, the whole catalogue of 23 design patterns can be presented with an intent catalogue of two pages.

REQUIREMENTS FOR A PATTERN AUTHORING AND READING ENVIRONMENT

From the discussion in the previous section, one can see that different approaches coexist in the pattern community to ease the reception of a pattern language. All these means share a common goal: explain the structure of a pattern language and make relations explicit. All can be expressed using labelled relations between patterns.

We argue to support the user by illustrating a combined set of relations in a pattern graph. The relations can be modelled as (labelled) edges between two pattern nodes *X* and *Y*. We propose the following relations:

- *X* uses *Y* in its solution.
- *X* is a variant of Pattern *Y.*
- *X* has a similar problem as *Y.*
- *X* is related in the related patterns section to Y.
- *X* specializes *Y* (in the sense of pattern inheritance).
- *X* connects to *Y* as part of the sequence S. In this case, the label includes *S* and a descriptive text that serves as the glue text in the sequence.
- *X* mentions *Y* in its context. This means that *Y* was applied before *Y.*

Besides these relations between patterns, we propose to add relations between patterns and other artefacts:

- *X* and *Y* are members of the same class or family. This relation is used to express classification schemes.
- *X* and *Y* involve a common participant *P*. In this case, *X* and *Y* are related to *P*. The label of the edge between the pattern and the participant describes the role of the participant in the pattern.
- *X* and *Y* can be found in the same known use *U.* This is modelled using an edge between the pattern and *U*. The label explains, how the pattern is used in the known use.

All these relations have been part of the textual pattern description. But especially the latter relations between

[1] since the book is not yet published, one can refer to [10] for a discussion on sequences from a PLoP perspective.

patterns and other artefacts were not made explicit (because neither the participants nor the known uses were considered as first class objects in current pattern structures). It is possible to generate the pattern graph from the patterns textual description.

Unfortunately, the number of relations may soon exceed the number of relations that can be visualized on the screen. It should thus be possible to filter relations so that only some of them are shown. For example by filtering all relations except the sequence relation, one can arrange the patterns in a linear sequence that can be read like traditional text.[2] Filtering all but the uses relations will result in an acyclic graph that can be visualized in a layered way. We refer to this combination of the pattern graph with the filtering algorithms as interactive pattern map.

Two more aspects can ease the usability of this interactive pattern map: Firstly automatic layout algorithms can help the user to detect clusters (e.g. classes of a classification scheme) and gain a better overview of the language. Secondly, the nodes should directly link to the textual form of the pattern. This means that the map serves as a navigation aid for the user.

Finally, from an author's point of view, it is very difficult to keep the textual pattern description and the interactive pattern map in synch. Changes of one representation should therefore automatically cause the change of the other representation.

In summary, a pattern authoring and reading environment should meet the following requirements:

(R1) Model patterns, artefacts, and relations in a patterns graph.

(R2) Provide interactive filtering mechanisms.

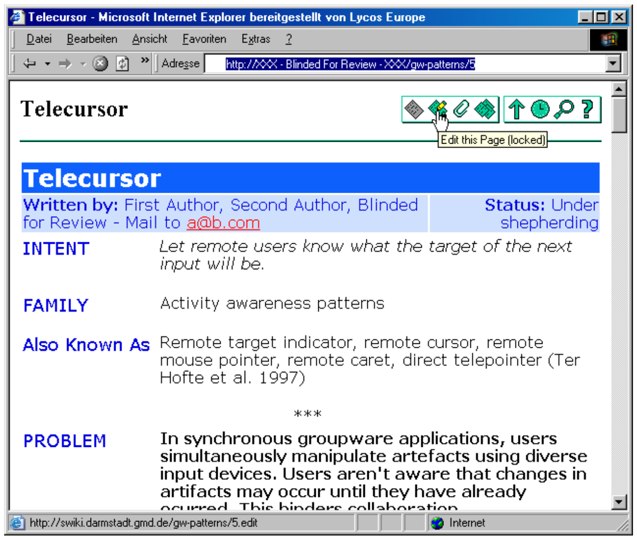
(R3) Make use of automatic layout algorithms.

(R4) Link the textual and the graphical representation of a pattern.

(R5) Automatically synchronize graphical and textual pattern representations.

A PROTOTYPICAL ENVIRONMENT

We are currently working on an authoring and reading environment, which meets all the requirements mentioned in the previous section.


Telecursor - Microsoft Internet Explorer bereitgestellt von Lycos Europe
Datei Bearbeiten Ansicht Favoriten Extras ?
Adresse http://XXX - Blinded For Review - XXX/gw-patterns/5
Edit this Page (locked)
http://swiki.darmstadt.gmd.de/gw-patterns/5.edit
Internet

Telecursor

Telecursor

Written by: First Author, Second Author, Blinded for Review - Mail to a@b.com | Status: Under shepherding

INTENT | *Let remote users know what the target of the next input will be.*

FAMILY | Activity awareness patterns

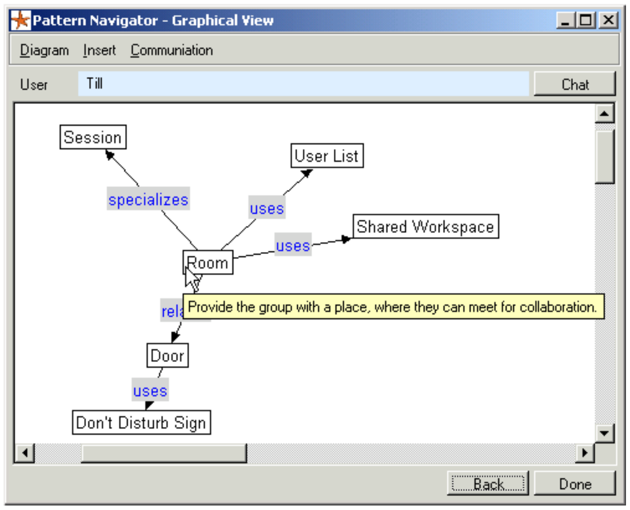
Also Known As | Remote target indicator, remote cursor, remote mouse pointer, remote caret, direct telepointer (Ter Hofte et al. 1997)

***

PROBLEM | In synchronous groupware applications, users simultaneously manipulate artefacts using diverse input devices. Users aren't aware that changes in artifacts may occur until they have already

Figure 1: The rendered textual version of the pattern.

It consists out of two parts: The textual authoring environment provides a form-based interface to enter the pattern itself. From our point of view, this part is the most important part of our environment since we still consider patterns as literature – although this literature is written in a very structured way. The authoring environment assists the author in linking patterns and formatting the textual output. An example of the textual output is given in figure 1.



Provide the group with a place, where they can meet for collaboration.

Figure 2: Prototype of the pattern navigator.

From the textual patterns, we generate a graphical representation in the pattern navigator (extracting some of the relations mentioned in the previous section). The pattern navigator shows all patterns as movable boxes. Relations between patterns are labeled directed edges. The navigator is based on the FUB Brainstorming system, which we used at our faculty to support collaborative brainstorming [9]. It allows the entering of new patterns and the manual connection between patterns and can be used by a group of users that is connected via the internet.

[2] This is more complicated if the sequence contains cycles.

All users can manipulate the map at the same time, which allows real-time collaboration (the system was implemented using the COAST groupware framework [14]). Figure 2 provides an example of the pattern navigator.

Currently, the mapping between patterns and the patterns map is only implemented in one direction – from patterns to the map. Thus, we don't yet support a full synchronization. Manipulations in the graphical version are not yet connected to the textual representation.

Requirements 2 and 4 are also currently under development. We hope to be able to show an integrated version of our environment at the CHI patterns workshop.

CONCLUSIONS

We have noted that the area of groupware development opens the field for a large collection of design patterns. Users need assistance when working with a large pattern catalogue. From other disciplines, we classified different relations that were used to provide such guidance.

We propose that the relations that we found in the literature combined with additional relations described in this paper should be used to create an interactive pattern map. To make this map comprehensible, tool support is needed. A pattern authoring and viewing environment should model the structure of the pattern language, provide filters, layout the language, link it to the textual representation and keep all visualizations in synch.

The collaborative pattern navigator that we showed in this paper is a first prototype for such an environment.

REFERENCES

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S.: A pattern language. New York: Oxford University Press, 1977.
2. Alexander, C.: The Nature of Order: The Process of Creating Life. Oxford University Press, 2003 (to appear).
3. Alexander, C.: The timeless way of building. New York: Oxford University Press, 1979.
4. Arnoldi, M., Beck, K., Bieri, M. and Lange, M.: Time Travel: A Pattern Language for Values That Change. In Dyso, P. and Devos, M. (Ed.): Proc. of *EuroPLoP 1999*, Konstanz, Germany, 2000, 121-136.
5. Bergin, J.: Coding at the Lowest Level - Coding Patterns for Java Beginners. In Rüping, A., Eckstein, J. and Schwanninger, C. (Ed.): *Proc. of EuroPLoP 2001*, Konstanz, Germany, 2002, 251-285.
6. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M.: A system of patterns. Chichester, West-Sussex, UK: John Wiley & Sons, 1996.
7. Fricke, A. and Völter, M.: Seminars - A Paedagogical Pattern Language about teaching seminars effectively. In Devos, M. and Rüping, A. (Ed.): *Proc. of EuroPLoP 2000*, Konstanz, Germany, 2001, 87-128.
8. Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.
9. Haake, J. M., Schümmer, T. and Haake, A.: Supporting Collaborative Exercises for Distance Education. *Proc. of HICSS 36*, IEEE: Hawaii, USA, 2003.
10. Harrison, N. B. and Coplien, J. O.: Pattern Sequences. In Rüping, A., Eckstein, J. and Schwanninger, C. (Ed.): *Proc. of EuroPLoP 2001*, Konstanz, Germany, 2002, 549-550.
11. Meira, N., e Silva, I. C. and Silva, A.: A Set of Agent Patterns for a More Expressive Approach. In Devos, M. and Rüping, A. (Ed.): *Proc. of EuroPLoP 2000*, Konstanz, Germany, 2001, 331-346.
12. Schmidt, D. C., Stal, M., Rohnert, H. and Buschmann, F.: Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects. Chichester, West-Sussex, UK: John Wiley & Sons, 2001.
13. Schümmer, T., Fernandez, A. and Holmer, T.: Groupware Patterns Homepage. http://www.groupware-patterns.org/, 2002.
14. Schümmer, T., Schümmer, J. and Schuckmann, C.: COAST - An Open Source Framework to Build Synchronous Groupware with Smalltalk. OpenCoast Development Group, 2001.
15. Tichy, W. F.: A Catalogue of General-Purpose Design Patterns. *Proc. of TOOLS 23*, IEEE Computer Society: 1998.
16. Völter, M.: Server-Side Components - A Pattern Language. In Rüping, A., Eckstein, J. and Schwanninger, C. (Ed.): *Proc. of EuroPLoP 2001*, Konstanz, Germany, 2002, 87-128.
17. Zimmer, W.: Relationships Between Design Patterns. In Coplien, J. O. and Schmidt, D. C. (Ed.): *Pattern Languages of Program Design*, Addison-Wesley: Reading, MA, 1995, 345-364.