Building a General Pattern Framework via Set Theory: Towards a Universal Pattern Approach

Alexander G. Mirnig, Manfred Tscheligi
Christian Doppler Laboratory for "Contextual Interfaces"
HCI & Usability Unit, ICT&S Center, University of Salzburg
Salzburg, Austria
Email: {firstname.lastname}@sbg.ac.at

***Abstract*— Patterns have been successfully employed for capturing knowledge about proven solutions to reoccurring problems in several domains. Despite that, there is still little literature available regarding pattern generation or common pattern quality standards across the various domains. We present an attempt for a universal (i.e., domain independent) pattern framework. Via basic set theory, it is possible to describe pattern sets that are composed of several subsets regarding pattern types, quantities, sequence, and other factors. We can thus describe patterns as sets of interrelated elements instead of isolated entities, thus corresponding with the scientific reality of complex problems with multiple relevant factors. The framework can be used to describe existing pattern languages and serve as a basis for new ones, regardless of the domain they are or were created for.**

Keywords-patterns; pattern basics; pattern framework; set theory

I. INTRODUCTION AND MOTIVATION

Patterns have been used as a tool for capturing knowledge about proven solutions to reoccurring problems in many domains. Most prominent among these domains are architecture and software design [1][2][6]. Patterns allow documenting knowledge about methods and practices in a structured and systematic manner. Another major benefit of patterns is that they can serve to "make implicit knowledge explicit" [10], i.e., they can be used to explicitly capture what is normally only acquired via experience after having worked in a certain field or domain for an extended period of time. The information contained in such patterns can then be provided to others (researchers or other interested parties) in a relatively quick and efficient manner. Despite this, there is little general (i.e., domain independent) literature available on patterns and pattern creation.

Having access to a structured collection of implicit and explicit knowledge about research practices is useful when conducting research in any domain. There is no *How to Generate Patterns in 10 Easy Steps* or similar basic literature. This is not an entirely new idea [8], and there has already been a big push in that direction by, e.g., the work of Meszaros and Doble [11] and Winn and Calder [13], which we want to expand and build upon.

Two of the main benefits of patterns are that they facilitate re-application of proven solutions and that they serve to make implicit knowledge explicit. These benefits are of particular importance to researches, who do not already have this knowledge themselves, i.e., it is a way to draw from a vast pool of knowledge. If working with patterns has extensive domain experience as a prerequisite, then those that would need that knowledge the most would benefit the least from it. The final goal of this research is to arrive at a structured but still easy to understand framework that captures the essence of patterns and makes them understandable as well as usable for practitioners and researchers in any domain. The first step is to provide a basic set theoretic analysis that allows to describe patterns and pattern languages in a general manner. This later on serves as a domain independent basis for reflections on how patterns can or should be created and structured.

We argue for a general strand of research on patterns as a means to capture knowledge about research practices. With such a theoretical basis available, practitioners from any domain could have a pool of knowledge to draw from, which would help them create patterns suitable for their needs. This should not mean that a variety in pattern languages and approaches is not desirable. It makes sense to assume that different domain requirements need different pattern approaches. However, the basics of patterns should ideally be similar for everyone and easily accessible, like with general mathematics. A statistician needs and employs different mathematical means than a fruit vendor. But both draw from the same pool of general mathematics as their basis. In our research, we take a step back, look at patterns from a general point of view and describe them via basic set theory [5]. A general analysis of patterns allows us to treat them as separate phenomena, independent of the domains they are created and used in. Set theory is one of the most basic, but at the same time very powerful, mathematical tools available. By using set theory, we can ensure consistency of our framework, while still keeping things basic and relatively easy to understand. An additional benefit of our approach is that it permits the creation of pattern sets across different pattern languages that address a similar purpose. This can facilitate the consolidation of already existing knowledge within the various domains. In this paper, we begin with an overview of existing general literature on patterns in Section II, followed by an outline of the initial proposed set theoretic pattern framework in Section III. In Section IV, we present our planned next steps for further iteration and finalization of

 ISBN: 978-1-61208-343-8

the framework and conclude with a few paragraphs on the perceived advantages and possible future challenges of our approach.

II. Related Work

Patterns have been employed in a multitude of application domains [1][6][12] and a good number of extensive pattern collections [3][4][7] have been created in the past. Literature on the pattern generation process itself, sometimes also referred as *pattern mining* [4], is still scarce [9]. Existing literature on pattern generation is mostly focused on specific domains [3][6][8][12]. The work of Gamma et al. [4] can be considered important elementary literature, but it is still centered on software design. Although covering a wide spectrum of software design problems, it is arguably of limited applicability outside of the software engineering domain. The same can be said about other specialized pattern generation guidances [8], which would require adaptation to be employed in other domains (e.g., biology or linguistics).

Meszaros and Doble [11], developed a pattern language for pattern writing, which serves to capture techniques and approaches that have been observed to be particularly effective at addressing certain reoccurring problems. Their *patterns for patterns* were divided into the following five sections: Context-Setting Patterns, Pattern Structuring Patterns, Pattern Naming and Referencing Patterns, Patterns for making Patterns Understandable, Pattern Language Structuring Patterns. Another interesting approach being quite similar in its aims to the one presented in this paper, is the *Pattern Language for Pattern Language Structure* by Winn and Calder [13]. They identified a common trait among pattern languages (i.e., they are symmetry breaking) and built a rough, nonformal general framework for pattern languages in multiple domains. These ideas are similar in concept to what we pursue in our research. The difference is that we want to provide a purely formal framework without or minimal statements regarding its content (such as types or traits). We want to focus on the basics behind patterns and structure these, so that they can be applied as widely as possible, although we intend to incorporate the work of Meszaros and Doble, and Winn and Calder at a later stage (see Section IV).

Another interesting aspect of patterns is that one single pattern is usually not enough to deal with a certain issue. Alexander et al. [2] already expressed this by stating the possibility of making buildings by "stringing together patterns". The pattern itself, however, does not include the information of which other pattern might be relevant in a particular case. This information is only available once the pattern is part of an actual pattern language. Borchers [3] introduced the notion of high level patterns, which reference lower level patterns to describe solutions to large scale design issues. This hierarchy is expressed via references in the patterns themselves, which is a good way of understanding and describing patterns as interconnected entities. A suitable framework for patterns and pattern languages should ideally be able to capture these relations between patterns.

III. The General Pattern Framework

A. Patterns and Pattern Sets

Before starting to build the framework, we first need to take a look at patterns, pattern languages, and the concepts behind them. Alexander [2] characterized patterns in the following way: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." So on a basic level, patterns can be understood as a structured assortment of statements. A pattern language is a complete hierarchy of patterns, ordered by their scope [12]. We will translate these concepts into a basic set theoretic structure by employing regular sets, ordered sets, and subsets, via the following example based on a Contextual User Experience (CUX) pattern structure by Krischkowsky et al. [8] (see Tab.1). Please note that this analysis would work for any pattern language that is or can be structured in a similar way, such as, e.g., the design patterns template laid out by Gamma et al. [6], but we wanted to give a more current and not software-centered example to prove our point.

TABLE I. CUX Pattern Structure [8]

Instructions on Each Pattern Section		
#	***Section Name***	***Instruction on Each Section***
1	Name	*The name of the pattern should shortly describe the suggestions for design by the pattern (2-3 words would be best).*
2	UX Factor	*List the UX factor(s) addressed within your chosen key finding (potential UX factors listed in this section can be e.g., workload, trust, fun/enjoyment, stress...). Please underpin your chosen UX factor(s) with a definition.*
3	Key Finding	*As short as possible - the best would be to describe your key finding (either from an empirical study or findings that are reported in literature) in one sentence.*
		. . .
8	Key-words	*Describe main topics addressed by the pattern in order to enable structured search.*
9	Sources	*Origin of the pattern (e.g., literature, other pattern, studies or results)*

We now want to generate an actual pattern language set, let us call it CUX Language (and refer to it as *CL* for brevity's sake), based on the structure outlined in Tab. 1. We can do so by introducing nine subsets (i.e., sets of the set *CL*) CL_1 to CL_9, each subset corresponding to one of the nine categories (from Name to Sources, respectively) described above. To actually generate a pattern for *CL*, we need to assign statements to each of the nine subsets. We do that by assigning a yet undefined set of statements *S* to *CL*, making

sure that none of the subsets remains empty. Note that 'statement' in this regard not only refers to full sentences, but also to single words or sequences of words which are not full sentences. We can generate n-number of *CL*-patterns P_1 to P_n this way.

Of course, simply arranging patterns into sets and subsets does not in itself guarantee that any of these patterns are actually useful or reasonable. What this analysis *can* tell us is (a) the pattern language (*CL*) the patterns are generated in, (b) how many statement categories a successful pattern generated in that language must contain, and (c) which statements can be found in which category, i.e., the patterns themselves. So, this elementary analysis has already yielded a powerful starting framework, via which we can express in a domain-independent manner how patterns and pattern languages stand in relation to each other, regardless of domain they were generated in.

B. Descriptors

CL does not yet fully qualify as a pattern language in the actual sense of the word and there are two things that an individual *CL*-pattern does not tell the reader at this stage. These are (a) which *other* patterns might be useful or even necessary for a given purpose, and (b) exactly at which point during a given task or activity and in which order will they be needed. Without knowing these, one can only guess what else they might need upon being presented with only a single pattern or depend on prior experience. It would be undesirable and arguably defeat the purpose of patterns, if extensive meta-knowledge were necessary to be able to use them successfully. This is why we enrich the basic set theoretic framework with specialized descriptor sets, which serve to understand patterns in context with each other. We shall again illustrate this via a simple example: Assume that we have three patterns, P_1 to P_3, which would help us in conducting a user study in the car. P_1 and P_2 are *CL*-patterns to reduce user distraction, whereas P_3 is a pattern about processing the data gained from the study. P_3 was created in a different pattern language, let us call that one *DL*. We can now specify which of these patterns we want or need and in which order by introducing an *ordered set* D. Let us further assume that we want to express that we need only one CL-pattern as well as the DL-pattern and that the DL-pattern will be needed after the CL-pattern. We can express all of this via the following example descriptor set D_1. Please note, that angle brackets ('<' and '>') denote an ordered set, as opposed to an ordinary set, which would be denoted by curly brackets ('{' and '}').

$$D_1: \langle CL^1, DL^1 \rangle \quad (1)$$

Instead, if we need both CL-patterns, we can express this via the following modification to D_1:

$$D_2: \langle CL^2, DL^1 \rangle \quad (2)$$

Considering the fact that D_1 in (1) does not tell us which of the two CL-patterns is needed, we could also specify a pattern directly, if not any of them would do:

$$D3: \langle P2, DL1 \rangle \quad (3)$$

But how do we now specify which of these descriptors (D_1-D_3) is the appropriate one for a given scenario? Patterns are created for a certain purpose, and in most cases that purpose is how to deal with a certain reoccurring problem. We can specify which descriptor fits a certain purpose better than another. To properly express this, we introduce the notion of targets *T* that contain the general purpose of a certain activity (e.g., car user experience). This is different from the problem-field of a pattern, since a given high level pattern could very well reference a lower level problem that addresses a different problem, while both serve the same general purpose. We can now map descriptors to targets, depending on what is needed. In this example, a target that does not require both CL-patterns would be assigned either D_1 or D_3, whereas one that does would be assigned D_2. So, in addition to being able to specifying the relations between patterns in a single pattern language, we are not confined to that single pattern language. This means, that we can also describe hierarchical pattern sets from different domains and pattern languages in the same framework. By adding one additional layer (targets and descriptors) to what was already available before, we have arrived at a highly modular and flexible pattern framework. Fig. 1 provides an overview of the interrelation of pattern languages, patterns, descriptors, and targets.

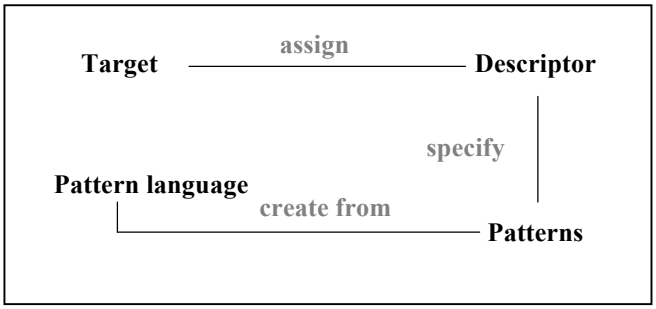

Figure 1. The Pattern Framework – Overview

Currently, one problem of this proposed structure is that any sets of statements can be made into a set and called a pattern language. This is hardly acceptable, of course, and needs to be rectified. We are currently still working on identifying requirements, that any set of statements should fulfill in order to be called a pattern language and what the respective descriptors should look like. Descriptors are ordered sets themselves, so each of their elements have a clearly defined spot in a concrete sequence. This rigid structure also means that it can be difficult to express that particular patterns could be relevant at any point in the sequence or at several fixed points. But it cannot be assumed that all patterns would only ever be relevant at one very specific point. Even if that were the case, it could similarly not be assumed that these specific points were always known. A refinement of the descriptor sets will be in order, to permit more numerous and less cumbersome expression possibilities with regard to sequences.

 ISBN: 978-1-61208-343-8

IV. NEXT STEPS

The framework outlined in this paper is still a work in progress. To arrive at a sufficiently detailed and refined framework, we are currently working on the following:

A. Refine the Framework: Pattern Languages, Descriptors, Targets

We intend to pursue a refinement of pattern language requirements similar to Winn and Calder's approach [13] and introduce symmetry breaking (or a similarly suitable property) as a necessary property of descriptors. We will then research the requirements a descriptor and its subsets need to fulfill in order to acquire that property. A more detailed analysis of pattern languages as hierarchical structures and how this translates into concrete descriptor sets is being worked on. Descriptors permit specifying patterns directly, via specifying the language they are part of, or with regard to other additional factors. The type of a pattern could be regarded as one such relevant factor. We will, therefore, incorporate the notion of pattern types into the framework, in particular the pattern types put forward by Meszaros and Doble [11]. We consider these as particularly important in this regard, due to their general nature, but also analyze domain-specific pattern types (e.g., the three types of software design patterns put forward by Gamma et al. [6]) will have to be taken into consideration. In addition, we will provide a more concrete structure for targets, with more detailed information on what a target is and the information it should contain.

B. Apply the Framework

Once the framework has been completed, we will demonstrate the suitability of the framework by actually generating and describing sample pattern sets from two very different domains (e.g., User Experience (UX) research and Neuroscience).

V. CONCLUSION

The great advantage of the approach described in this paper is that patterns are separate from descriptors, which are themselves separate from the targets. This means that patterns can be generated as usual, descriptors generated and assigned on an as-needed basis. For the pattern user, this means that they do not have to scour vast databases of patterns for those they might need. All they need is to have a look at the descriptor(s) that is/are assigned to the target they have in mind. Thus, existing pattern databases can be expanded with descriptors, which help make them more usable and reduce the amount of domain experience and previous knowledge required in order to employ patterns successfully.

But even more importantly, descriptors can specify patterns with regard to certain properties, such as pattern language, context, etc. Descriptors functions similarly to references are contained in the patterns themselves (as suggested by Borchers [3]), but enable additional or alternative references to other patterns at any time, since they are not actual parts of a pattern. This means that descriptors can be used to describe virtually any pattern set, regardless of which domain(s) its patterns came from or when the pattern was created. Not only is it possible to capture the hierarchical order of existing pattern languages via descriptors, but also reference patterns from other languages that might fit a certain purpose. This means that the framework is not tied to a single pattern language or even a single domain and permits references to patterns from multiple pattern languages. We therefore consider it a suitable basis for domain independent pattern research.

ACKNOWLEDGMENT

We gratefully acknowledge the financial support by the Austrian Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development (Christian Doppler Laboratory for „Contextual Interfaces").

REFERENCES

[1] C. Alexander, The Timeless Way of Building, New York: Oxford University Press, 1979.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, A Pattern Language: Towns, Buildings, Construction, Oxford: University Press, 1979.

[3] J. Borchers, A pattern approach to interaction design, New York: John Wiley & Sons, 2001.

[4] A. Dearden and J. Finlay, "Pattern Languages in HCI: A Critical Review," HCI, Volume 21, January 2006, pp. 49-102.

[5] K. Devlin, The Joy of Sets: fundamentals of contemporary set theory, 2nd ed., Springer, 1993.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software, Boston: Addison-Wesley Professional, 1995.

[7] S. Günther and T. Cleenewerck, "Design principles for internal domain-specific languages: a pattern catalog illustrated by Ruby," Proc. 17th Conf. on Pattern Languages of Programs (PLOP '10). ACM, New York, NY, USA, , Article 3 , pp. 1-35, DOI=10.1145/2493288.2493291, retrieved: April, 2014.

[8] A. Krischkowsky, D. Wurhofer, N. Perterer, and M. Tscheligi, "Developing Patterns Step-by-Step: A Pattern Generation Guidance for HCI Researchers," Proc. PATTERNS 2013, The Fifth International Conferences on Pervasive Patterns and Applications, ThinkMind Digital Library, Valencia, Spain, May 2013, pp. 66–72.

[9] D. Martin, T. Rodden, M. Rouncefield, I.Sommerville, and S. Viller, "Finding Patterns in the Fieldwork," Proc. Seventh European Conf. on Computer-Supported Cooperative Work, Bonn, Germany, September 2001, pp. 39-58.

[10] D. May and P. Taylor, "Knowledge management with patterns," Commun. ACM 46, 7, July 2003, pp. 94-99, DOI=10.1145/792704.792705, retrieved: April, 2014.

[11] G. Meszaros and J. Doble, "A pattern language for pattern writing," Pattern languages of program design 3, Robert C. Martin, Dirk Riehle, and Frank Buschmann (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, November 1997, pp. 529-574.

[12] J. Tidwell, "Designing Interfaces : Patterns for Effective Interaction Design," O'Reilly Media, Inc., 2005.

[13] T. Winn and P. Calder, "A pattern language for pattern language structure," Proc. 2002 Conf. on Pattern Languages of Programs - Volume 13 (CRPIT '02), James Noble (Ed.), Vol. 13. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, June 2003, pp. 45-58.

 ISBN: 978-1-61208-343-8