A Survey of Existing Approaches for Pattern Search and Selection

Aliaksandr Birukou
DISI - University of Trento, Italy

1. INTRODUCTION

Given the steadily growing number of patterns in the literature and online repositories, it is hard to be aware of the patterns that exist and to select patterns appropriate to the problem at hand. In this paper, we present an overview of existing approaches related to pattern search and selection.

Fifteen years ago, Gamma et al. (also known as GoF) stated the problem of selecting patterns: “With more than 20 design patterns in the catalog to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalog is new and unfamiliar to you” [Gamma et al. 1995]. Since then, the problem of selecting patterns has become more critical, as the number of documented patterns has been continually increasing, adding to the selection problem also the problem of searching patterns from which to select. For instance, Rising’s *Pattern Almanac* [Rising 2000] lists more than 1200 patterns. In the past ten years since the publication of the almanac, many new patterns and books on patterns have been published. In 2007 Henninger and Correa counted 2241 patterns focusing solely on those related to software [Henninger and Corrêa 2007].

In the domains containing more than ten patterns, the problem of choosing the appropriate pattern is particularly hard to solve for inexperienced programmers, as illustrated by the following quote [Sommerville 2004]:

> Only experienced software engineers who have a deep knowledge of patterns can use them effectively. These developers can recognize generic situations where a pattern can be applied. Inexperienced programmers, even if they have read the pattern books, will always find it hard to decide whether they can reuse a pattern or need to develop a special-purpose solution.

We define the problems of searching and selecting patterns as follows:

—How to find patterns that can solve a particular problem?

—How to select the pattern to apply among the available ones?

Note that we intentionally do not focus on *design problems*, but, rather, on general problems, thereby considering any kind of patterns as long as they fit the definition of pattern by Alexander [1977]:

> "[...] 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."

Therefore, examples of considered patterns include but not limited to organizational, pedagogical and business patterns, or patterns for organizing conferences [Haskins 2006], conducting meetings [Haase and Miedl 2007;

Schuemmer and Tandler 2007], building online communities [Homsky and Raveh 2007], etc. For the very same reason we use “apply a pattern” through the paper, and not “implement a pattern”, since the latter is more specific to code.

By *searching* patterns we mean getting information about existing patterns, no matter where this information is coming from: the literature or the Internet. By *selecting* patterns we mean choosing a pattern in the list of patterns, where the list is obtained at the stage of searching patterns or pre-defined (e.g., in a narrow domain that only contains 9 patterns instead of searching one should look through all 9 patterns). These problem definition are motivated by and consistent with the descriptions of the problems of search and selection of patterns in the works of Henninger and Correa [2007], Weiss and Mouratidis [2008], Greene et al. [2003], Yoshioka et al. [2008] and Deng et al. [2005].

We present an overview of existing approaches for pattern search and selection. Some of the approaches reviewed in the paper were found by searching Google Scholar with the keywords "pattern search", "pattern selection", "design pattern search", "design pattern selection", "software pattern search", "software pattern selection" and by reviewing the first 100 of results. Also, we have reviewed past EuroPLoP, PLoP, and other *PLoP proceedings. Some of the papers were cited in the reviewed papers. However, the most of the approaches were known to the author already because of his previous work on pattern selection. We believe that the reviewed approaches cover most of the work in the field of pattern search and selection.

This paper might be of interest to several groups of people. The primary audience for this paper are developers of pattern repositories and pattern retrieval systems that are willing to enhance search and selection of patterns in their systems. People developing pattern languages can benefit from this paper by choosing a format of the pattern language in such a way that is is easier to provide selection tools for this language. For instance, they could provide additional information required for the selection, or list forces and consequences explicitly. Conference organizers can use reviewed approaches to provide search for related patterns to the patter writers. Other interested groups include researchers on the application of patterns as well as people dealing with search and selection in different domains (e.g., selecting products from an online catalog).

This paper has the following structure: Section 2 defines the problems of pattern searching and selection. Section 3 introduces dimensions of analysis of approaches for pattern searching and selection. Those dimensions are used to classify the existing approaches in Section 4. Section 5 discusses strengths and shortcomings of existing approaches and outlines the directions for the future work, and Section 6 concludes the paper.

2. THE PROBLEMS OF PATTERN SEARCHING AND SELECTION

In this section we describe the problems of pattern searching and selection in detail. Figure 1 shows a simplified overview of the process of applying a pattern to solve a problem. A person (developer, designer, etc.) starts with the problem, which is formulated as a problem description. Then, at the optional step of *searching patterns*, the problem description is used as a query to search for patterns that can potentially solve the problem. This step can be skipped if not supported by the tool at hand (e.g., not all pattern catalogues support search). As the next step, the person *selects a pattern* from the list of found/existing patterns. The final step is *applying the pattern* in order to solve the problem. Note that in this paper we do not focus on the step of applying the pattern. The process can be repeated several times, until the desired state (to have the original problem solved with the selected pattern) is reached. A real process of applying patterns for solving problems can involve more steps, transitions, and feedback loops. An interested reader can find more detailed descriptions of the process of applying patterns in [Wang et al. 2005; Yoshioka et al. 2008].

2.1 Pattern searching

The problem of searching patterns is that of getting information about existing patterns. This information can come from literature, Internet, word-of-mouth communication such as interaction with colleagues and friends.

The problem of searching patterns presents the following challenges:

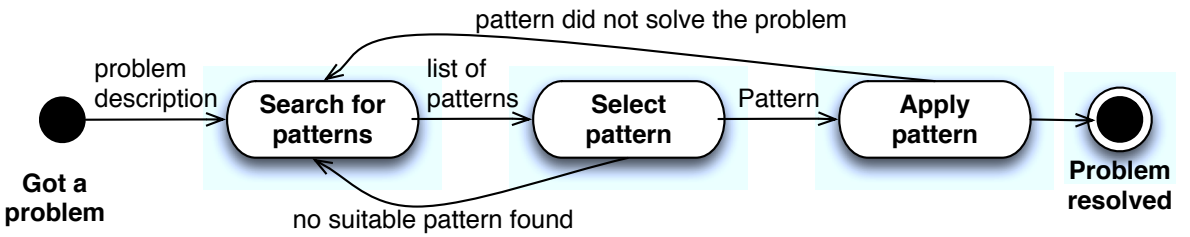

Fig. 1. The process of applying patterns for solving problems

—Not all pattern descriptions are available on the Internet. For instance, in the study by Henninger and Correa [2007] 31% of software-related[2] patterns were inaccessible via the Web and were only available in book format (proceedings, journal, book).

—There is no single repository of pattern. Henninger and Correa counted 121 software-related pattern collections (i.e., loosely coupled patterns located in a common location such as a repository, paper, book, website) that contained more than one pattern.

—There is a variety of pattern description forms (those used in GoF [Gamma et al. 1995] or Pattern Oriented Software Architecture (POSA) [Buschmann et al. 1996] books, Pattern Language Markup Language (PLML) [Fincher 2003] or PLMLx [3], to name a few). This makes harder the comparison of the patterns, for instance, to determine if two patterns are the same, different, or versions of the same pattern.

—There are no standard mechanisms for indexing, accessing, and inter-linking pattern catalogs.

—There is no pattern search engine that covers most of the available patterns and is universally accepted by the community, i.e. there is no Google for patterns.

—It is not possible to search using the form of the pattern, e.g., find a specific string in the problem and another string in the trade-offs description of the pattern.

2.2 Pattern selection

The problem of selecting a pattern is choosing a pattern to apply from a list of patterns. The list of patterns is either given, e.g., comes from a catalogue, or, as shown in Figure 1, is obtained as the result of searching patterns. Note that in this paper we do not consider a more complex problem of selecting a pattern sequence, leaving this for future work.

The problem of selecting patterns presents the following challenges:

—The selection is context-sensitive. For instance, in case of selecting a software pattern, the selection depends on the application and the set of patterns already in place: what can be the right pattern to use in one system might be inappropriate in another.

—The list obtained as the result of the search can contain a lot of patterns in different forms.

—It might be hard to choose the right pattern if there are duplicates and variants (slightly modified versions of each other) among patterns [Henninger and Corrêa 2007].

[2]Henninger and Correa focused on those patterns related to software development and the software development process, including topics such as software project management.

[3]Extended Pattern Language Markup Language. `http://www.cs.kent.ac.uk/people/staff/saf/patterns/diethelm/plmlx_doc/plml_doc.dtd.html`

—The list of patterns normally contains patterns which are solving not the problem at hand, but another, may be similar, problem. Therefore, an analysis must be carried out first to check if the pattern addresses the right problem.

—There are positive and negative consequences of applying a pattern. Therefore, some analysis may be needed to estimate the consequences and see if the pattern is worth applying.

Depending on the domain of patterns, there can be other, more specific challenges in their selection. For instance, for design patterns, non-functional requirements satisfied by patterns are mostly expressed as informal text [Gross and Yu 2001], and therefore, it is hard to see if the pattern matches the problem specification.

2.3 Specifics of pattern search and selection

The problem of pattern search and selection is different from the general problem of search and selection of relevant information as considered in Information Retrieval (IR)(see, e.g., [2006] for an example of the latter). Pattern search and selection is different from IR because: (i) patterns are structured documents where different parts express very different types of information, while in IR the documents are considered as a bag of words or a set of weighted terms; (ii) patterns are often linked to each other in a pattern language or via user-created relations, while the documents in IR are usually considered independently; and (iii) patterns accumulate the knowledge of experts in dealing with problems, therefore it might be hard to find mapping between the language of patterns and the language of inexperienced people facing the problems those patterns solve. Due to these reasons, pattern search and selection requires specialized tools.

3. DIMENSIONS OF ANALYSIS

In this section, we introduce dimensions, which we use in Section 4 for the analysis of existing approaches. We found it is convenient to split them into *features*, which are characteristics of the approach that support users in solving the problems of searching and selection of patterns, and *properties*, which are characteristics of the approach that are not directly related to the considered problems.

Note that several features are taken from the overview of existing User Interface (UI) pattern collections by [Deng et al. 2005, p.7] and we refer readers interested in the comparison of pattern repositories to their work, which provides a more detailed analysis of UI collections. The scope of the analysis in this paper is broader since it goes beyond pattern collections. Therefore, some details mentioned in [Deng et al. 2005] and specific to pattern collections, such as import/export of patterns to the collection, are not relevant in our analysis.

Table I lists the features, grouped in those supporting search, selection, and both.

We have considered the following features relevant to the problem of pattern search:

—**Crawling:** finding new links to patterns automatically on the Internet.

—**Indexing:** crawling is usually followed by indexing, which makes the content of the pattern searchable.

—**Searching:** how users can search patterns: using tags and keywords, searching the full text of pattern descriptions, possibility to specify where in pattern description the keyword must be present, searching using additional data, such as requirements, properties, quality goals addressed by patterns.

We have considered the following features relevant to the problem of pattern selection:

—**Support for selection:** how the approach supports users who need to select a pattern among several available alternatives. Possible forms of such support are: **ranked list of results**, which provides the ranked list of results returned to a query; **relevance score**, which shows a relevance score of a pattern w.r.t. query; **algorithm/method**, which is applied for selecting patterns.

—**Recommendation facilities:** providing suggestions on which pattern to choose for the problem at hand.

We have considered the following features relevant to the both problems of pattern search and selection:

Table I. Features of pattern search and selection approaches

Search	
Crawling	
Indexing	
Searching	tag/keyword search
	full text search
	form-aware search
	search based on additional data
Selection	
Support for selection	ranked list of results
	relevance score
	algorithm/method
Recommendation facilities	
Search and Selection	
Browsing	display pattern list
	view pattern details
Navigation	
Linking	within repository
	with patterns in another repository
Collaboration	users can add or edit patterns
	users can add relationships
	users can tag patterns
	users can comment/annotate patterns
	users can rate patterns

Table II. Properties of pattern search and selection approaches

Domain	
Requires pre-processing of patterns	no
	yes, manual
	yes, automated
Pattern format	human-readable
	machine-processable
	customizable
Method	
Tool support	
Accessibility	
Availability	
Interoperability	

—**Browsing:** this includes features **display pattern list**, which shows the list of available patterns, and **view pattern details**, which shows the complete description of a pattern.

—**Navigation:** going from one pattern to another via links or relations between them.

—**Linking:** possibility to specify the links or relations between patterns within the repository or with patterns from other repositories.

—**Collaboration:** features that allow users to collaboratively add and edit patterns and relations between patterns, tag, annotate, comment on, rate patterns and share their experience with using patterns.

Table II lists the properties of the approaches for pattern search and selection.

We have defined the following properties that are important for characterizing approaches, but are not falling in the category of end-user features:

—**Domain:** the subset of patterns considered by the approach. Examples include: HCI, software, security, business, analysis, architecture[4].

—**Requires pre-processing of patterns:** if it is necessary to pre-process the collection of patterns before applying the approach. Required pre-processing can be manual or automated.

—**Pattern format:** if it is possible to get pattern description in human-readable and machine-processable formats and whether it is possible to customize the pattern template (fields contained in a pattern).

—**Method:** does the approach provide a method, i.e., a systematic and ordered process for pattern search or selection?

—**Tool support:** was the approach implemented in a tool or is it purely theoretical, i.e. provides the set of written guidelines?

—**Accessibility:** how the tool delivered (via Internet, a desktop application).

—**Availability:** Is it possible to download the tool or access it on the Web?

—**Interoperability:** possibility to access the tool or repository via some API.

4. APPROACHES FOR PATTERN SEARCH AND SELECTION

In this section we review different approaches to pattern selection. They are grouped in several clusters, as illustrated in Table III.

4.1 Pattern repositories and catalogs

Recently, there have been several efforts in making patterns available in online pattern repositories, where they can be browsed and searched using various criteria. An early example was the Pattern Almanac [Rising 2000], which is also available in electronic form[5]. More recent examples are the PatternShare[6] site hosted by Microsoft between 2006-2007, Yahoo Design Pattern Library[7], Sun collection of J2EE patterns[8], and computer-mediated interaction patterns[9]. In this section, we review the most recent approaches approaches for selecting patterns stored in such repositories. For a more complete overview of pattern collections and pattern repositories we refer the reader to the survey papers by Deng et al. [2005] and by Henninger and Correa [2007].

To the best of our knowledge, most of the repositories created in the past remain oblivious to the advent of Web 2.0 and list content defined by the repository creator without the possibility of collaborative editing, tagging, bookmarking and other social features. The sad thing about this is no matter how heavily the repository is used for searching patterns, it does not change and improve over time, if not maintained. However, several wiki-based repositories such as PatternForge[10] or Open Pattern Repository[11] were created recently trying to overcome such shortcomings and to use the power of the community in order to enrich repositories with tags, links and other user-generated content.

In order to store patterns in a repository, a structured pattern representation must be adopted. There have been several proposals for structural pattern representation, including the PLML, PLMLx, Mackenzie-Nickull Architectural Patterns Meta Model [MacKenzie and Nickull 2005], and Entity Meta-Specification Language (EML) [Welicki et al. 2007].

---

[4]Note that domains may overlap, they are provided here as specified in the papers or tool websites, or as deduced from the available material (if not explicitly mentioned)

[5]`http://www.smallmemory.com/almanac`

[6]`patternshare.org`

[7]`http://developer.yahoo.com/ypatterns/`

[8]`http://java.sun.com/blueprints/patterns/`

[9]`http://www.cmi-patterns.org/`

[10]`http://www.patternforge.net/wiki/index.php?title=Main_Page`

[11]`http://patternrepository.org/`

Table III. Approaches for pattern search and selection

		Approaches																		
		pattern repositories				recommendation systems			formal languages								search engines		other approaches	
		Greene	OPR	PatternForge	PLaNet	REBUILDER	BARPC	IC-Patterns	Hinojosa	Pearson	PatternsBox	GRL	GRL-Prolog	Gross	Wang	Zdun	PatternSeer	Google Design	ArchPad	ESSDP
SEARCH																				
Crawling		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-
Indexing		-	+	+	+	-	-	+	-	-	-	-	+	-	-	-	+	+	-	-
Searching	tag/keyword search	-	+	+	+	-	-	+	-	-	-	-	-	-	-	-	+	-	-	-
	full text search	+	+	+	+	-	-	+	-	-	-	-	-	-	-	-	+	+	-	-
	form-aware search	+	?	-	+	-	-	+	-	-	-	-	-	-	-	-	?	-	-	-
	search based on additional data	+	+	-	-	+	-	+	-	-	-	-	+	-	-	-	?	-	-	-
SELECTION																				
Support for selection	ranked list of results	-	+	+	+	+	+	+	-	+	-	-	+	-	+	-	+	+	-	+
	relevance score	-	?	-	-	-	+	-	-	+	-	-	+	-	+	-	?	-	-	+
	algorithm/method	decision trees	?	wiki	wiki	CBR	clustering, collaborative filtering	IR, CBR, ICF	logical inference (Progol)	rules	navigation over pattern repository	formal language	formal language, rules, reasoning	semi-formal language	formal language	formal pattern grammar, QOC	?	based on PageRank	decision-based design method	rule-based expert system
Recommendation facilities		+	?	-	-	+	+	+	-	+	-	-	-	-	-	-	-	-	-	+
SEARCH and SELECTION																				
Browsing	display pattern list	+	+	+	+	+	-	+	-	-	+	+	?	-	+	+	+	+	+	-
	view pattern details	+	+	+	+	-	-	+	+	-	-	+	?	-	-	-	+	+	-	-
Navigation		+	+	+	+	-	-	+	-	-	+	-	-	-	-	-	?	-	-	-
Linking	within repository	+	+	+	+	-	-	+	-	-	+	+	+	-	+	+	-	-	-	-
	with patterns in another repository	-	?	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Collaboration	users can add/edit patterns	+	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	users can add relations	?	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	users can tag patterns	-	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	users can comment and annotate patterns	-	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	users can rate patterns	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Properties																				
Domain		HCI	Software	Any	eLearning	Design	Requirement Analysis	Security	Design	Software	Design	Architecture	Security	Design	Design	Software	Any	Design	Architecture	Design
Requires pre-processing of patterns	no	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	-	-
	yes, manual	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	-	-	+	+
	yes, automated	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-
Pattern format	human-readable	+	+	+	+	+		+			+	+	+	+	+	+	+	+	+	+
	machine-processable	+	+	-	+	+	+	+	+	+	+	-	+	-	+	+	-	-	-	+
	customizable	+	+	+	?	-		+			-	-	-	-	-	+	-	-	-	-
Method		-	-	-	-	+	+	-	+	+	+	+	+	-	+	+	-	-	+	+
Tool support		+	+	+	+	+	-	+	-	+	+	-	+	-	-	-	+	+	-	+
Accessibility		web	web	web	web	CASE	?	web	-	?	?	N/A	?	N/A	N/A	N/A	web	web	N/A	web
Availability		-	?	+	+	-	-	-	-	-	-	N/A	-	N/A	N/A	N/A	-	+	N/A	-
Interoperability		-	REST API	-	HTML API	-	-	-	-	-	+	N/A	-	N/A	N/A	N/A	?	-	N/A	-

Weiss and Birukou [2007] created a wiki-based pattern repository called PatternForge. The wiki organizes short descriptions of patterns called pattlets in collections, and allowed for their tagging, linking to other pattlets in a pattern language. Since the approach uses wiki, many useful features, such as navigation, linking, tagging, commenting, searching for patterns come for free from the wiki or corresponding plugins. Unlike other approaches, PatternForge aimed at supporting any kind of patterns as long as a pattlet description can be provided. Even though the repository failed to build the community around it, PatternForge is still available on the Web.

The Pattern Language Network (PLaNet)[12] developed a pattern repository for people using Web 2.0 in assessment, learning and teaching, to capture and share their examples of good practice as patterns. The repository is based on wiki, and therefore allows for easy search, comments, tagging, and linking patterns. It also provides possibility to filter the list of patterns based on the author, the status of the pattern (seed idea, alpha, beta) or the author's confidence in the pattern (from 0 to 3). The repository is available online even though the PLaNet project ended, however, the most recent pattern was added on the 2d of March 2009.

Open Pattern Repository (OPR) is an ongoing project that aims at creating a publicly available and freely usable online repository for patterns. The focus is on architectural and design software patterns, but authors also envision storing there information about software technologies (frameworks, APIs, etc.) in a pattern form. The repository supports different pattern forms, adding, editing and removal of patterns by users, browsing by categories, search of patterns matching certain criteria, and relationships between patterns (e.g., variants of a pattern). The repository is accessible via REST interface and will soon be publicly available.

Greene et al. [2003] describe a tool that supports pattern-assisted design and development in HCI. The tool provides means for creating patterns in standard or user-defined form, browsing them, navigating from one pattern to another via links. Patterns should be entered in the repository where they are stored as XML documents and can be visualized in human-readable form. Patterns can be searched using their full text or by querying subsets of the fields of the patterns. They can be also annotated with user-defined properties, which can be also used for the search. The tool also provides a mechanism for helping users to find patterns "answering" their questions. The mechanism is based on decision trees and does not require manual intervention for providing recommendations.

4.2 Recommendation systems

Several approaches exploit past user experience in order to suggest suitable patterns. The REBUILDER [Gomes et al. 2002] framework adopts a Case-Based Reasoning (CBR) approach, where cases represent situations in which a pattern was applied in the past to a software design. REBUILDER supports the retrieval and adaptation of patterns. Cases are described in terms of UML class diagrams. Cases are retrieved based on a combination of structural similarity between the current design and a pattern, as well as the semantic distance between class names and role names in the pattern.

Birukou and Weiss [2009] developed a multi-agent system for recommending patterns, called IC-Patterns. In the system the patterns are selected using the Implicit Culture Framework (ICF), which recommends patterns based on past user actions. The system also supports conventional IR (Information Retrieval) and CBR methods for finding relevant patterns. IC-Patterns consists of a web-based user interface on the client side and a multi-agent platform on the server side. A user accesses the system by submitting a query that includes a description of the problem and a description of the project where the problem is encountered. The problem is described by a set of keywords, optionally restricted to specific elements of the pattern description, e.g. problem, context. The project description can be represented as a set of properties such as project size, required level of data protection, etc. The prototype of IC-Patterns works for the security patterns, but the system can support different domains and forms of patterns. An indexing of patterns for the IR search is preformed automatically using the Lucene

[12] http://www.planetspace.org.uk/

IR library[13]. However, for the use of form-aware search a machine-processable structured pattern description is required.

Shu-Hang et al. [2007] propose a pattern selection method based on a pattern clustering analysis algorithm and a collaborative filtering recommendation algorithm. The method deals with requirement analysis patterns for e-Business applications. In order to apply the clustering and recommendation algorithms, one must first provide formal descriptions of project situations and patterns. Then, the method calculates pattern contribution coefficient, pattern similarity degree, and correlation between patterns and project situations, based on the provided formal descriptions. The calculated metrics are used for clustering and recommendations. It is hard to estimate the effort required for providing formal descriptions of all patterns and project situations, and it is have to be done manually. However, the paper reports on the evaluation where recommendations were compared with the decision of software experts and proved to be useful.

4.3 Formal languages

This section describes approaches that use (semi-)formal languages and grammar for pattern representation and base selection on such representation.

Hinojosa [2004] developed a cognitive model of the process software engineers employ to select a specific implementation for a system design. The approach deals with the design patterns from Gamma et al. [1995] classified under the category of behavioral patterns, because those patterns have most implementation strategies. To achieve the goal, the patterns were mapped to a set of features consumed by the reasoning engine, based on the Progol language. The author used the data about real world implementation decisions in order to infer which features guided engineers to a specific implementation. As the result, for each of the behavioral design patterns, the model provided a set of relevant features for each pattern. The approach by Hinojosa is interesting in the sense that it tries to provide a cognitive model for human reasoning during the selection of patterns. However, it is hardly scalable and applicable in different domains, since all patterns must be manually transformed into sets of predicates and decisions on pattern selections must be processed manually.

Albin-Amiot et al. [2001] developed the Patterns-Box tool, which provides assistance in designing a software architecture. The tool supports the choice of the right design pattern depending on the application context. The tool operates with a repository of design patterns, where each design pattern must be first expressed in a formal Pattern Description Language (PDL). During the selection, a formal model of the current application context is used in order to find the most suitable pattern in the pattern repository. Patterns-Box also supports HTML navigation between patterns and is integrated with another tool developed by authors.

Pearson and Shen [2010] describe a decision support system that recommends design patterns satisfying contextual requirements such as privacy, security, etc. The system takes context requirements of the required design as an input of a rule-based engine to trigger decisions about patterns that could be appropriate to use within that context. The context requirements are elicited by a questionnaire. The approach is targeted at non-expert developers and architects. The system deals with design patterns, which are parametrized based on the selection criteria. The system does not depend on the format of the patterns as long as they are connected with the rules. For each domain, rules and patterns must be created based upon industry practices in the domain. In other words, the system developed by Pearson and Shen is a kind of an expert system which operates on top of expert knowledge represented by the rules.

Mussbacher et al. [2006] present a goal-oriented requirement language (GRL) that formalizes the forces of patterns and relations between patterns. The formalization and explicit representation of forces complements conventional textual descriptions of the patterns and enables trade-off analysis of forces during the selection of a suitable pattern. the formalization are GRL graphs that capture pattern forces and can be used to assess the qualitative impact of various solutions to a functional goal. Moreover, the authors also provide several graphs that

[13] `http://lucene.apache.org/java/docs/index.html`

formalize the relationships between forces of patterns and non-functional requirements common in the domain of architecture design. Since the approach also formalizes connections between patterns in a pattern language, it also supports the selection of combinations of patterns. The approach at the moment did not provide any tool supporting the selection.

Weiss and Mouratidis [2008] apply GRL in the domain of security patterns. They go a couple of steps further than Mussbacher et al. [2006] and apply Prolog reasoning on top of GRL representations of patterns and provide a pattern search engine, supporting the selection of the security patterns. A user submits security and other non-functional requirements as a query to the search engine, which returns the list of patterns fulfilling the specified requirements. Furthermore, the approach supports the search for a combination of security patterns that meet a specified set of requirements.

Zdun [2007] proposes an approach for software pattern selection based on desired quality attributes and pattern relations. The approach requires formalizing the pattern relationships in a pattern language grammar and annotation of the patterns with their effects on quality goals. This step is performed manually. From the pattern language grammar pattern sequences are automatically derived and used for narrowing down the search space to related patterns, thus decreasing the time spent for the evaluation of alternatives. In addition to the automatic derivation of pattern sequences, the author proposes the concept of design spaces (an explicit representation of alternative design options and reasons for choosing between those options) for documenting design decisions. The design decisions are subsequently used for question-option-criteria (QOC) analysis of the design space, thereby allowing one to obtain a detailed decision map for the design decision at hand. The author suggests that the QOC analysis should only be performed if the information in the annotated pattern language grammar is not sufficient for making an informed design decision.

Gross and Yu [2001] propose adding an explicit specification of non-functional requirements (NFRs) to the description of design patterns and to use them for pattern selection. Their approach is semi-formal and requires annotation of patterns with their contribution to NFRs and than manual analysis of the applicability of a pattern in a certain context.

Wang et al. [2005] present a method for selecting design patterns that fulfill the non-functional requirements of the architectural design. The method first uses a non-functional requirements framework (NFR) to retrieve a prioritized list of patterns that might be suitable for a given set of requirements. Each pattern in the list is then decomposed into components with AND or OR relationships hierarchically. The traceability from the components of software architectural design to the components of design patterns is analyzed. Finally, the applicability of each design pattern is determined using the NFR.

4.4 Search engines

There are several search engines for patterns. PatternSeer[14] aimed at delivering a system that crawls and indexes pattern descriptions on the Internet and makes them accessible to users via keyword-based search. Recently, Google provided a custom search engine[15] indexing several online pattern repositories. The problem with current solutions is their limited coverage of patterns. This reminds one of the problems early Internet had – just eleven years ago it was better to use several search engines to get more different results for a query.

4.5 Other approaches

Zimmermann et al. [2008] propose a decision-based design method, ArchPad, for domain-specific pattern selection. The method is based on architectural decisions identified in the requirement models and consists of the four refinement stages: 1) executive decisions, requirement analysis; 2) conceptual decisions including selection of architectural patterns and key technology choices; 3) detailed technology decisions, design patterns as architecture

[14] Developed by Ademar Aguiar, now inactive

[15] Design pattern search. `http://www.google.com/coop/cse?cx=000531763273211731096\%3Ab-lv61obcte`

alternatives; 4) vendor asset level decisions and selection of implementation, deployment and test patterns. As a consequence, ArchPad supports several types of patterns as architectural decision alternatives: analysis and architectural patterns at the executive and conceptual level, design patterns on the technology level, and implementation and test patterns at the vendor asset layer. For each domain, ArchPad requires the creation of an RADM, a Reusable Architectural Decision Model, which is made available like a pattern catalog or language. Each RADM is created manually by experts and contains patterns that proved to be applicable in the past projects in the domain. RADMs are thus artifacts that contain domain-specific decision models guiding practitioners through pattern selection and application. Patterns in each RADM have to be updated over time, especially the patterns related to implementation decisions must be adapted as technology matures.

Even though RADMs have to be created manually for each domain, they will prove useful in case there is a number of projects in the same domain. Due to the analytical nature of the approach, the tool support is not required, but can be provided in the future. One of the strengths of the method is that it originates from the analysis of authors' decision making and pattern selection practices in industry projects. The method goes beyond pattern selection in the sense that it supports architecture decisions in general, not only those related to pattern selection.

Kung et al. [2003] propose a methodology for constructing expert systems for suggesting design patterns to solve problems faced by designers. They have implemented a prototype, Expert System for Suggesting Design Patterns (ESSDP), which selects design patterns based on the user's requirements. A user interacts with the system in a question-answer manner, which helps to narrow down the selection process. At the end of the interaction, a suitable design pattern is offered to the user. The ESSDP system assumes the knowledge acquisition as the primary step of the methodology. At this step, human experts must fill in the knowledge base with some pre-defined rules. One of the limitations of the system is that it only supports rule-based knowledge bases.

5. DISCUSSION

From Table III we can see that existing approaches can be clustered into the following categories:

—**Pattern repositories and catalogs** provide support for storing, searching, browsing, navigation, and collaborative tagging, linking, and annotation of patterns.

—**Search engines** provide crawling and search functionalities and do not store pattern descriptions, but just index them.

—**Recommendation systems** provide suggestions on which pattern(s) could be applicable in a certain situation.

—**Formal approaches** propose (semi-)formal languages and frameworks for pattern representation and tools for reasoning on such representations.

—**Other approaches**.

Existing approaches that support pattern users in the search and selection of patterns have several shortcomings:

(1) they usually require additional effort during the authoring and selection of patterns. For instance, authors need specify metadata about their patterns or formalize them in a certain language;

(2) once created, pattern repositories require effort in maintaining and updating information;

(3) most of the solutions are domain-specific with no relations to patterns from another domain or repository. For instance, many solutions deal with design patterns proposed by GoF [Gamma et al. 1995], while there are also security or HCI patterns that can be used together with design patterns;

(4) they rarely support collaboration and personalized recommendations. Thus, as [May and Taylor 2003] say, much of the information how patterns are selected by users remains tacit despite the existence of these tools.

(5) most of the tools are not available after the end of the project, or even when the paper is published. Greater availability of the tools, especially if they are open source, can enable code reuse and easier comparison between the tools.

From the observations above, we can suggest the following directions in the research on pattern search and selection:

(1) **Pattern search engines**. Since PatternSeer is inactive, pattern repositories and Google Design Pattern Search only index pre-defined collections of patterns, there is no pattern search engine that automatically crawls pattern descriptions on the Internet. Henninger and Correa [Henninger and Corrêa 2007] provide some insights on how this can be done. It is a challenging task, but the potential success can be huge, since the pattern search engine is likely to be adopted by the pattern community.

(2) **Pattern sequences**. Existing approaches do not support automated collection and analysis of pattern sequences, i.e., ways how patterns are used together in an application. Currently, it is only possible to do such collection by asking users to annotate the design. However, such collection might become possible with the spread of Model-Driven Software Development paradigm.

(3) **Interoperability**. Existing approaches rarely integrate with other tools or provide an open API. However, such APIs can enable the development of cross-collection pattern search tools.

(4) **Non-intrusiveness**. The approaches should be less intrusive, i.e., more integrated with the conventional workflow of the user. In some approaches, the amount of work for implementing the approach is roughly the same as manual selection of patterns by experts would require. Wiki-based approaches sound promising in this direction.

6. CONCLUSION

In this paper we have defined problems of pattern search and selection and reviewed existing approaches for solving the problem. The approaches have been classified according to features and properties they provide.

In this survey we focused on different domains of patterns, including but not limited to software, HCI, business, organizational patterns. Related, but very different problems of selecting architectural primitives [Zdun and Avgeriou 2008] and selecting pattern sequences have not been considered. Those are possible directions for future research.

7. ACKNOWLEDGEMENT

I am deeply indebted to my shepherd Uwe van Heesch and to all participants of Workshop A at EuroPLoP 2010 for their insightful, constructive, and encouraging comments. This work has been partly supported by the European Union FP7 projects COMPAS and LiquidPublication, grants no. 215175 and 213360.

REFERENCES

ALBIN-AMIOT, H., COINTE, P., GUEHENEUC, Y.-G., AND JUSSIEN, N. 2001. Instantiating and detecting design patterns: Putting bits and pieces together. In *International Conference on Automated Software Engineering*. 166–173.

ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)* Later printing Ed. Oxford University Press.

BIRUKOU, A. 2009. *Implicit Culture Framework for behavior transfer*. VDM Verlag.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., AND STAL, M. 1996. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns* 1 Ed. Wiley.

DENG, J., KEMP, E., AND TODD, E. G. 2005. Managing UI pattern collections. In *ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction*. ACM, New York, NY, USA, 31–38.

FAN, W., GORDON, M., AND PATHAK, P. 2006. On linear mixture of expert approaches to information retrieval. *Decision Support Systems 42,* 2, 975–987.

FINCHER, S. 2003. PLML: Pattern language markup language. report of workshop held at CHI, Interfaces, 56 (pp. 26-28). Tech. rep.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc.

GOMES, P., PEREIRA, F. C., PAIVA, P., SECO, N., CARREIRO, P., FERREIRA, J. L., AND BENTO, C. 2002. Using CBR for automation of software design patterns. In *European Conference on Advances in Case-Based Reasoning*. Springer-Verlag, London, UK, 534–548.

GREENE, S. L., MATCHEN, P. M., JONES, L., THOMAS, J. C., AND CALLERY, M. 2003. Tool-based decision support for pattern assisted development. In *CHI 2003 workshop on HCI Patterns: Concepts and Tools*.

GROSS, D. AND YU, E. S. K. 2001. From non-functional requirements to design through patterns. *Requirements Engineering 6,* 1, 18–36.

HAASE, M. AND MIEDL, M. 2007. Patterns for leading effective and efficient meetings. part two. In *Proceedings of EuroPLoP'07*.

HANG, G. S., QING, L. Y., ZHONG, J. M., JING, G., AND JUAN, L. H. 2007. A Requirement Analysis Pattern Selection Method for E-Business Project Situation. In *Proceedings of the IEEE International Conference on e-Business Engineering*. ICEBE '07. IEEE Computer Society, Washington, DC, USA, 347–350.

HASKINS, C. 2006. "not just another conference". pattern language for conducting a successful niche conference. In *Proceedings of VikingPLoP'06*.

HENNINGER, S. AND CORRÊA, V. 2007. Software pattern communities: current practices and challenges. In *PLOP '07: Proceedings of the 14th Conference on Pattern Languages of Programs*. ACM, New York, NY, USA, 1–19.

HINOJOSA, A. 2004. A cognitive model of design pattern selection.

HOMSKY, O. AND RAVEH, A. 2007. Pattern language for online communities. In *Proceedings of EuroPLoP 2007: 12th European Conference on Pattern Languages of Programs*.

KUNG, D. C., BHAMBHANI, H., SHAH, R., AND PANCHOLI, G. 2003. An expert system for suggesting design patterns: a methodology and a prototype. In *Software Engineering With Computational Intelligence*, T. M. Khoshgoftaar, Ed. Series in Engineering and Computer Science. Kluwer International.

MACKENZIE, M. AND NICKULL, D. 2005. Mackenzie-Nickull architectural patterns meta model. Tech. rep., Adobe Systems.

MAY, D. AND TAYLOR, P. 2003. Knowledge management with patterns. *Communications of the ACM 46*, 94–99.

MUSSBACHER, G., WEISS, M., AND AMYOT, D. 2006. Formalizing architectural patterns with the Goal-oriented Requirement Language. In *Nordic Pattern Languages of Programs Conference*.

PEARSON, S. AND SHEN, Y. 2010. Context-aware privacy design pattern selection. In *Proceedings of the 7th international conference on Trust, privacy and security in digital business*. TrustBus'10. Springer-Verlag, Berlin, Heidelberg, 69–80.

RISING, L. 2000. *The Pattern Almanac*. Addison-Wesley Longman Publishing Co., Inc.

SCHUEMMER, T. AND TANDLER, P. 2007. Patterns for technology enhanced meetings. In *Proceedings of EuroPLoP'07*.

SOMMERVILLE, I. 2004. *Software engineering* 7th Ed. Addison-Wesley, Boston, MA, USA.

WANG, J., SONG, Y.-T., AND CHUNG, L. 2005. From software architecture to design patterns: A case study of an NFR approach. In *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and ACIS International Workshop on Self-Assembling Wireless Networks*. IEEE Computer Society, Washington, DC, USA, 170–177.

WEISS, M. AND BIRUKOU, A. 2007. Building a pattern repository: Benefitting from the open, lightweight, and participative nature of wikis. In *Workshop on Wikis for Software Engineering at ACM WikiSym, 2007 International Symposium on Wikis (WikiSym), Montre'al, Que'bec, Canada, October 21-23*.

WEISS, M. AND MOURATIDIS, H. 2008. Selecting security patterns that fulfill security requirements. In *International Conference on Requirements Engineering*. 169–172.

WELICKI, L., LOVELLE, J. M. C., AND AGUILAR, L. J. 2007. Patterns meta-specification and cataloging: Towards a more dynamic patterns life cycle. In *International Workshop on Software Patterns: Addressing Challenges* at *COMPSAC 2007*.

YOSHIOKA, N., WASHIZAKI, H., AND MARUYAMA, K. 2008. A survey on security patterns. *Progress in Informatics 5*, 35–47.

ZDUN, U. 2007. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience 37,* 9, 983–1016.

ZDUN, U. AND AVGERIOU, P. 2008. A catalog of architectural primitives for modeling architectural patterns. *Inf. Softw. Technol. 50,* 9-10, 1003–1034.

ZIMMERMANN, O., ZDUN, U., GSCHWIND, T., AND LEYMANN, F. 2008. Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method. In Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008). *Software Architecture, Working IEEE/IFIP Conference on 0*, 157–166.