Human-Computer Interaction Design Patterns: Structure, Methods, and Tools

Christian Kruschitz
Department of Informatics-Systems
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
chris@isys.uni-klu.ac.at

Martin Hitz
Department of Informatics-Systems
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
hitz@isys.uni-klu.ac.at

Abstract—**Design patterns play an important role when managing design knowledge for later reuse. In the Human-Computer Interaction (HCI) community, design patterns are an often used tool for sharing design knowledge among user interface (UI) designers as well as non UI experts. An HCI design pattern consists of several different components. The first component is the *structure* of a pattern, which encapsulates the description of the problem, its context, and the solution suggested by the pattern. *Relationships* and *semantics* are important when design patterns are used in *pattern management tools*. To make sure that the developed patterns satisfy their users, it is important to *evaluate* and *validate* the patterns' content.**

Keywords – HCI patterns, History, Organization, Evaluation, Validation, Standardization .

I. Introduction

This paper is an extended version of [41] (PATTERNS09), and gives an in-depth overview on the literature and research on Human-Computer Interaction **(**HCI) design patterns.

HCI design patterns are an important tool for knowledge sharing in the domain of Human-Computer Interaction. To avoid reinventing the wheel again and again, design patterns identify and document best practice solutions to support user interface designers in their daily work in order to improve their productivity and make the design process more efficient.

Over the past years, many research activities in the area of design patterns have aimed to make them easier to use. The research focused on the pattern structure, organizing principles, semantics, relationships, evaluation of the usefulness of patterns, and tool support. This paper gives an overview of the above-mentioned topics.

We start with a historical overview of design patterns from the birth of the pattern concept to today's activities in the community. In Section III, we provide definitions of relevant terms. Section IV deals with the pattern structures from the early beginnings in architectural design to pattern forms, which are currently used by the HCI design pattern community.

The following sections deal with research topics on design patterns, starting with the *Organizing Principles* which are focusing on the categorization schemes of design patterns for easier retrieval of the right pattern for a given design problem within an collection or pattern language. Section VI shows how to identify relationships among design patterns. Relationships represent a key concept to gain the full reuse potential from individual patterns. Proper consideration of relationships promises even more powerful search and navigation opportunities. Section VII describes research approaches on how to enrich design patterns with semantic information. By using ontologies, it is possible to share HCI design patterns across different collections and it is easier to identify patterns for a specific design problem. When using patterns in interface design it is important that the used pattern is valid for the problem to be solved. Therefore, Section VIII presents some approaches of how to evaluate and validate HCI design patterns. Section IX introduces some software tools, which have been developed in the past years. The last section deals with standardization approaches.

II. History

Christopher Alexander, architect and mathematician, first talked about patterns in his PhD thesis which was subsequently published as the book "Notes on the Synthesis of Form" in 1964 (see timeline in Figure 1). Christopher Alexander laid the cornerstone of the later well known concept of design patterns. Alexander argues that design problems are getting more and more complex so that they exceed the designer's abilities to come up with a solution from scratch. Furthermore, problems cannot show its own solution, only a set of requirements, when combined together, the requirements create a new idea [4]. From 1975 to 1979 Alexander published several books on the concept of design patterns and pattern languages [3][5][6]. Although his concept was originally meant to support reuse of architectural design knowledge, it found its way into the HCI community where it was first mentioned 1986 by Donald Norman and Stephen Draper [51].

Ward Cunningham and Kent Beck have adopted this principle to object-oriented programming (OOP) and user interface (UI) implementation in 1987 [8][55]. They presented five patterns for designing window-based user interfaces in Smalltalk:

- WindowPerTask
- FewPanes
- StandardPanes

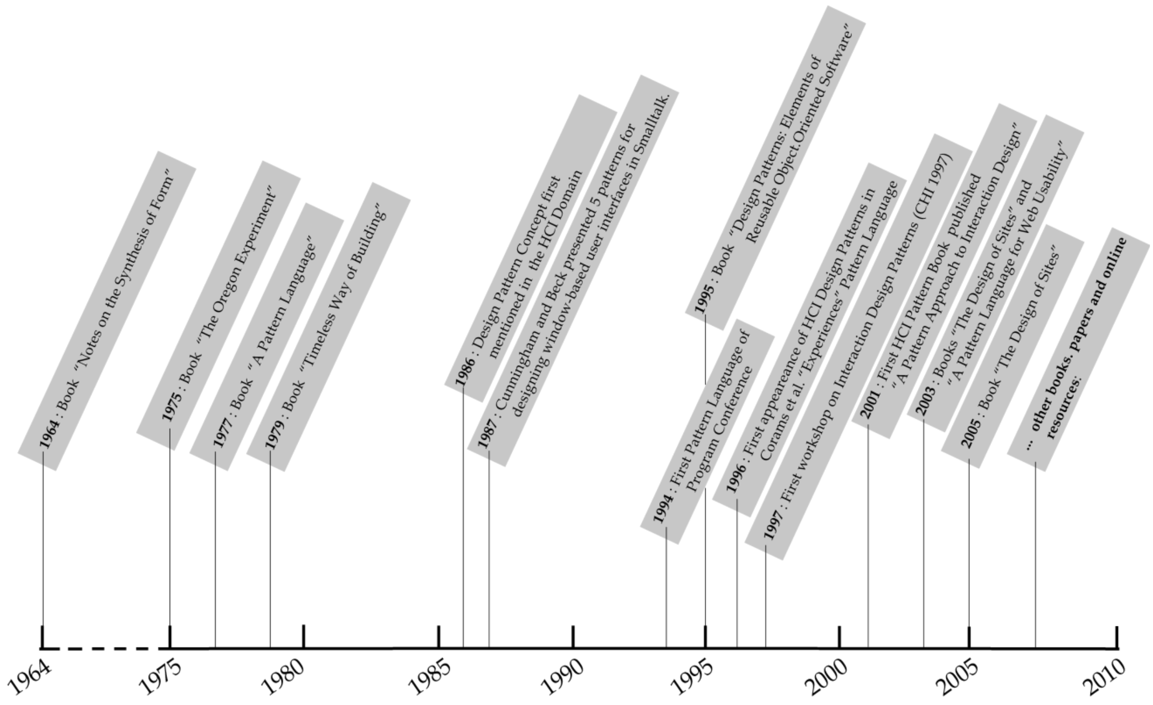

1964 : Book "Notes on the Synthesis of Form"
1975 : Book "The Oregon Experiment"
1977 : Book "A Pattern Language"
1979 : Book "Timeless Way of Building"
1986 : Design Pattern Concept first mentioned in the HCI Domain
1987 : Cunningham and Beck presented 5 patterns for designing window-based user interfaces in Smalltalk.
1995 : Book "Design Patterns: Elements of Reusable Object.Oriented Software"
1994 : First Pattern Language of Program Conference
1996 : First appeareance of HCI Design Patterns in Corams et al. "Experiences" Pattern Language
1997 : First workshop on Interaction Design Patterns (CHI 1997)
2001 : First HCI Pattern Book published "A Pattern Approach to Interaction Design"
2003 : Books "The Design of Sites" and "A Pattern Language for Web Usability"
2005 : Book "The Design of Sites"
... other books, papers and online resources.
1964
1975
1980
1985
1990
1995
2000
2005
2010

Figure 1: HCI Design Patterns Main Activities Timeline

- NounsAndVerbs
- ShortMenus

This small "pattern language" was intended to give novices in Smalltalk the possibility to use the language with all its strengths and avoid pitfalls. Cunningham and Beck were surprised of the good interfaces their users designed.

The Hillside Group, which now sponsors pattern conferences all over the world, has organized the first PLoP (Pattern Languages of Programs) conference in October 1994 with 80 participants.

Design patterns made their breakthrough in the software engineering community when Erich Gamma et al. published one of the bestselling books in software engineering, "Design Patterns: Elements of Reusable Object-Oriented Software" [25]. This book was awarded in the Journal of Object Oriented Programming (September 1995 Issue) "the best OO book of 1995" and "the best OO book of all times". From this time on many design patterns and pattern languages in software engineering as well as in user interface engineering have been published.

In HCI, the actual start of the design pattern era was 1996 when Tod Coram and Jim Lee published the first design patterns of a pattern language for user-centered interface design [14]. Their intention was to provide high level patterns with which user interface designers could build graphical user interfaces which are pleasurable and productive to use. In 1997 the first CHI workshop on pattern languages in user interface design was organized. The participants explored the use of a pattern language in user interface design to make HCI knowledge reusable in different applications [7]. At this time, user interface toolkits have emerged to support user interface designers and software engineers. However, the workshop participants stressed that a more general description of user interface design know-how, which is detached from a specific implementation platform, would be desirable and agreed that design patterns could be an appropriate tool. Design patterns reside on a higher level of abstraction than UI toolkits and are not bound to source-code for a specific implementation of the addressed problem. Furthermore, patterns are written in such a general way that they give pattern users the possibility to decide how specific widgets should be arranged to concretize the patterns' solutions.

Other pattern workshops focusing on HCI design patterns followed (see Table 1). Beside the discussion about the concept of design patterns in the HCI domain in workshops around the world, several books were published addressing design patterns and pattern languages for the HCI domain:

- "A Pattern Approach to Interactive Design" provides design patterns for interactive exhibits and user interface design [11].
- "The Design of Sites" is a comprehensive pattern language to help developing customer-centered websites [18].
- "A Pattern Language of Web Usability" provides a pattern language for the design of usable websites [27].
- "Designing Interfaces" is a collection of HCI design patterns which addresses how to build desktop and mobile user interfaces [60].
- "Patterns for Computer-Mediated Interaction" provides patterns for the design of Human-Computer-Human Interaction (HCHI) [56].

- “Designing Social Interfaces” written by the curator of the Yahoo!Design Pattern Library. This book provides patterns for designing a usable social website [15].
- “User-Centered Interaction Design Patterns for Interactive Digital Television Applications” shows how television applications can be designed based on design patterns [42].

Conference	Year	Title	Ref.
INTERACT	1999	Usability Pattern Languages	[28]
ChiliPLoP	1999	CHI Meets PLoP	[9]
CHI	2000	Pattern Languages for Interaction Design: Building Momentum	[24]
CHI	2001	Patterns: What’s in it for HCI (Panel)	[10]
CHI	2003	Perspectives on HCI Patterns	[22]

Table 1: Workshops on HCI Design Patterns

Beside the publication of books, the World Wide Web is the perfect medium to disseminate and publish HCI design patterns across the HCI community. Over time, many repositories were published, some have disappeared and others are still a point of reference to UI designers. Some of them are listed below with a short comment on each of them. For a more detailed description of a selected set of the mentioned Web repositories, cf. Section VIII.

- “Common Ground” is Jennifer Tidwell’s pattern language for HCI design [59].
- “Designing Interfaces” is the companion website to the same named book [60].
- “Little Springs Design – Mobile UI Design Resources” provides a design pattern collection for designing UIs for mobile devices [45].
- “UI Patterns – User Interface Design Pattern Library”, describes design patterns for desktop and mobile phone UI design [61].
- “Yahoo!Design Pattern Library” is a very popular design pattern collection by Yahoo! [66].
- “Welie.com – Patterns in Interaction Design” is a huge design pattern repository which addresses patterns for Desktop- and Webdesign [64].
- “Portland Pattern Repository” maybe the oldest pattern repository [53].

Beside the above mentioned design pattern repositories there exists several design pattern portals providing a collection of references to design pattern resources. These are:

- “The Interaction Design Patterns Page”, a collection of links to interaction design pattern resources [19].
- “hcipatterns.org”, provides information to HCI design patterns web resources, books and other related stuff like papers [29].
- “The Pattern Gallery”, a listing of design pattern forms with a short statement [23].
- “The Hillside Group”, is the organization who organizes the PLoP conferences. A good resource to start with the design pattern concept [58].

In the following section, we define the terms *HCI design pattern*, *design pattern catalogue/collection*, and *pattern languages*. Furthermore, we describe the components of HCI design patterns (see Figure 2), and describe the developments of the last years.

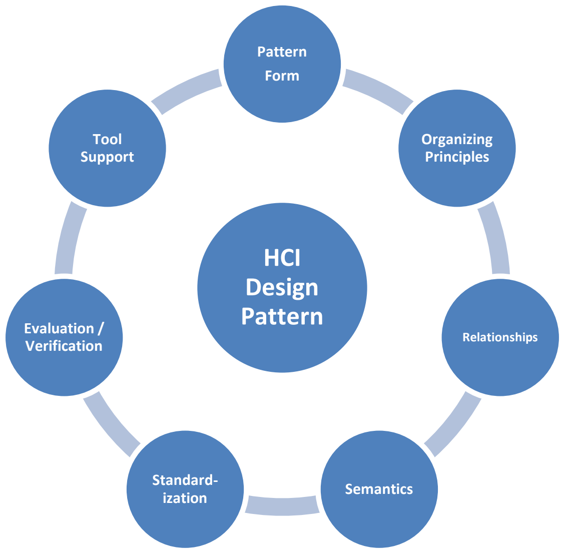


Figure 2: Components of HCI Design Patterns

III. DEFINITIONS

A. *HCI Design Pattern*

An HCI design pattern describes a recurring problem together with a proven solution. An HCI design pattern, in the following referred to as “pattern” or “design pattern”, has a well-defined form, which is dependent on the individual author’s preferences. A pattern form should be used consistently across a pattern language or pattern collection. This makes it easier for pattern users to understand the problem, context, and solution of a pattern throughout a pattern collection / language. The pattern itself, when it is part of a collection or a pattern language, may have references to other patterns.

B. *Design Pattern Catalogue / Collection*

Patterns are stored in design pattern catalogues or collections. The patterns in such a catalogue are categorized to support faster navigation within the repository. In this case, patterns show almost no relationships among each other and thus do not form a fully interconnected system. Instead several patterns stand more or less alone and have no or few connections to predecessor or successor patterns. Furthermore, such a collection usually does not completely cover a specific application domain.

C. *Pattern Language*

In contrast to a pattern catalogue / collection, a pattern language is a *complete* set of patterns for a given family of design problems in a given domain. A pattern language describes problems by means of high-level design patterns, which may be solved by lower-level design patterns. The design patterns are connected through relationships, so that they constitute a network.

In a pattern language, the "words" are the patterns, while the connections between patterns represent the "rules of grammar" which are situated in the pattern itself. When words and rules of grammar are combined, a "sentence" is generated. Sentences can be built in many different forms when the rules of grammar are followed. So there is not only one path through a pattern language, it offers several possibilities to solve a design problem. A good example is "The Design of Sites" by van Duyne et al., a pattern language that allows designers to articulate an infinite variety of Web designs [18]. Figure 3 visualizes a part of a pattern language with focus on online shopping [62].

IV. Pattern Form

Patterns are written by researchers and UI designers in a well-defined format, the so-called pattern form. This form is dependent on the author's preferences but several canonical forms have been established in the history of design patterns. These are described below in more detail.

A. Alexandrian Form

Christopher Alexander has invented the concept of design patterns as a problem / solution pair and presented them in a common format [3], which consists of:

- **Picture:** Shows an archetypal example of the pattern in use.
- **Introductory Paragraph:** This sets the pattern in the context of other, larger scale patterns.
- **Headline:** A short description of the problem.
- **Body:** Detailed description of the problem.
- **Solution:** The solution of the pattern which is written as a design instruction.
- **Diagram:** Sketches the solution in the form of a diagram.
- **Closing Paragraph:** Gives references to other patterns and describes how this pattern relates to with other, smaller patterns.

This pattern form is used with minor changes by Todd Coram and Jim Lee [14], Jan Borchers [11], Ian Graham [27], Mark Irons [37], Douglas van Duyne et al. [18], and Eric Chung et al. [13].

B. Software Engineering Design Patterns

There are influential approaches stemming from the software engineering domain, which are briefly described below:

The Gang of Four Form

This form is used in the book "Design Patterns: Elements of Reusable Object-Oriented Software" [25] and for many other OO software design patterns from different authors.

- **Pattern Name and Classification**: The pattern name describes in a word or two what the pattern is about and the classification groups the pattern with similar problems.
- **Intent**: A short statement what the pattern does and some words about its rationale and intent.
- **Also Known As**: Alternative names for the pattern.
- **Motivation**: A scenario how the class and object structures solve the addressed problem.
- **Applicability:** Describes the situation in which the design pattern can be applied.
- **Participants**: Addresses which classes and/or objects participate in the design pattern.
- **Collaborations**: Represents how the participants collaborate with each other.
- **Consequences**: Tells the user how the pattern supports its objectives.
- **Implementation**: Describes how to implement the pattern and how to overcome common pitfalls.
- **Sample Code**: Contains some code fragments on how to implement the pattern.
- **Known uses**: Examples of implementations proving the value of the pattern.
- **Related Patterns**: References to other patterns which are closely related.

The Portland Form

The Portland Pattern Form [53] is not as clearly structured as the others. The patterns are structured as text paragraphs. Ward Cunningham describes the form he uses in the Portland Pattern Repository as follows:

> *"Each pattern in the Portland Form makes a statement that goes something like: 'such and so forces create this or that problem, therefore, build a thing-a-ma-jig to deal with them.' The pattern takes its name from the thing-a-ma-jig, the solution. Each pattern in the Portland Form also places itself and the forces that create it within the context of other forces, both stronger and weaker, and the solutions they require. A wise designer resolves the stronger forces first, then goes on to address weaker ones. Patterns capture this ordering by citing stronger and weaker patterns in opening and closing paragraphs."*

The Coplien Form

James Coplien used the so-called Canonical Form to describe his patterns. This form is also called *Coplien Form* because he was one of the more famous pattern writers in the early stages of the software patterns movement [1].

- **Name**: Describes the name of the pattern.
- **Problem**: Addresses which problem will be solved by the pattern.
- **Context**: Tells the user in which context the pattern can be applied.

- **Forces**: This element describes (possibly conflicting) requirements and their impact on the design pattern.
- **Solution**: Shows the user how to balance the forces and solve the problems.
- **Resulting Context**: States which context is generated by applying the pattern.
- **Rationale**: Describes why the solution is implemented in such a way.
- **Author**: Name of author and creation date.

Software engineers have adapted the Alexandrian Form to describe their patterns. Significant changes are the introduction of the content elements – *Implementation*, *Sample Code* and *Participant*s, which describe OO programming facets like source code and UML diagrams.

C. HCI Design Pattern Forms

UI PATTERN FORM

This pattern form was developed at the INTERACT patterns workshop in 1999 [28]. It comprises seven content elements:

- **Name**: Shortly describes the pattern's intent.
- **Sensitizing Example**: This component should sensitize the reader to the application of the pattern. It is usually a screenshot or drawing of the pattern's solution.
- **Problem Statement**: Describes the conflicts (trade-offs) between "forces" guiding the design approach.
- **Body:** Textual description of the pattern's intent.
- **Solution Statement**: Tells what to do (and not how to do it).
- **Technical Representation:** This example solution is more detailed and intended to inform HCI experts about the pattern's solution.
- **Related Patterns:** References to successor patterns which enhance or are similar to the pattern.

TIDWELL FORM

Jenifer Tidwell is using a very minimalistic form, which is used throughout her book "Designing Interfaces" and the accompanying website [60].

- **Name:** Describes the pattern's intention and defines a unique reference number.
- **Sensitizing Image:** This image sensitizes the reader to the pattern's solution.
- **What:** Short problem statement.
- **Use When:** Describes the context in which this pattern can be used.
- **Why:** Describes the design rationale.
- **How:** Represents the solution part of the pattern.
- **Examples:** Screenshots of the instantiated pattern with a short description.

HCI design pattern authors are not using content elements such as *Implementation* or *Source Code* for their patterns. It is not necessary to provide source code for demonstration purposes of the pattern's solution. The problem is that interaction principles are implemented in many different programming languages. Therefore, the pattern is written in a more abstract way than software patterns. The pattern form is significantly stronger based on the Alexandrian Form.

More pattern forms, which were recently used, can be found at Sally Fincher's portal [23].

V. ORGANIZING PRINCIPLES

Alexander has organized his pattern language into levels of physical scale. He starts with high-level patterns which describe the size and distribution of towns and proceeds in several steps to low-level patterns which describe individual rooms [3].

TASKS	INFORMATION INTERACTION
Retrieval	Retrieval tasks have (static) information passing from the artefact to the user(s). The flow is usually initiated by the user(s).
Monitoring	Monitoring tasks have (dynamic) information passing from the artefact to the user(s). The information may come from 'beyond' the artefact. The flow is usually initiated by the artefact.
Controlling	Controlling tasks have information passing from the artefact to the user(s) and a separate flow from the user(s) to the artefact. The flow may be initiated by either the user(s) – proactive control – or by the artefact – reactive control
Construction	Construction tasks have the user(s) putting new information into the artefact
Transaction	Transaction tasks have the user(s) putting linked changes into the artefact. They are often accompanied by a corresponding change in the outside world.
Modification	Modification tasks have user(s) changing information already in the system. They may be modifying 'attribute values' or 'structure'
"Calculation"	Calculation tasks have the user(s) putting information into the system which it then transforms and passes back to the users (not necessarily synchronous).
Workflow	Workflow tasks have the system providing information to the user(s) which they then transforms and passes back to the system (not necessarily synchronous).
Communication	Communication tasks have one group of users putting information into the system that it passes to another group of users.

Table2: Organizing Principle by Fincher and Windsor [21]

In analogy, an organizing principle for HCI patterns, as Fincher and Windsor mentioned, should allow users to find patterns they need within a large repository. An organizing principle should meet at least the following objectives (cited from [24]):

- ***Taxonomise*** *– It must allow finding and selecting material from a large repository.*
- ***Proximate*** *– It must allow users to locate supporting, perhaps inter-related, patterns applicable to their solution.*

- ***Evaluative*** *– The problem should be considered from different viewpoints. So that it is possible to evaluate and change the users approach or to confirm the quality of their existing solution.*
- ***Generative*** *– It would be advantageous to support users to consider the problem from different points of view and allow for building new solutions, which have not been previously considered.*

Fincher and Windsor have adapted the Alexandrian structure of scale to UI design, starting with a high-level category *Society*, and descending via *System*, *Application*, *UI Structure* and *Component* to the low-level categories *Primitive* and *Physical Detail*. However, they do not consider this categorization sufficient for UI designers to find a pattern for their problem. So they suggested a second and a third structure. The second one is based on the design-by-type-of-task, where they have defined tasks based on the information flow, which includes categorizations such as *Task-Retrieval*, *-Monitoring*, *-Controlling*, *-Construction* and others (see Table 2). The reason for the last categorization structure is as the authors stated in their article: "*It is as common, as 'natural', for UI designers to structure their design not around the nature of the interaction (the 'how'), but the stuff that is to be interacted with (the 'what')*". So they suggested another category to satisfy UI designers which comprises categories such as *Volume*, *Complexity*, *Structure*, and *Dynamics*. *Structure* is further subdivided into *amorphous*, *sequential*, *hierarchical*, *directed acyclic graph*, and *web*. *Dynamics* is subdivided into *creation / termination*, *rate of change* and *patterns of change*.

Another approach has been put forward by Mahemoff and Johnston [46]: UI patterns can be assigned to four different categories. First, the *Task* category comprises all patterns addressing actions users might perform. Second, the *User Profile* category gathers patterns focusing on user groups. Third, *User-Interface Elements* helps designers and programmers to understand when to use a specific interface element or widget. Finally, *Entire System* patterns capture the issues of specific kinds of systems.

Van Welie and van der Veer [62] are organizing their patterns by means of "scaling the problem". As design is considered a top-down activity, their categorization is top-down as well. Problems are scaled from high-level problems like *Business Goals* to more detailed problems like *Task Level* and *Action Level* as shown in Figure 3. Another possibility to scale or group design patterns suggested by van Welie and van der Veer are to organize them according to their *Function* or to *Problem Similarity*, where *Function* can be subdivided into *Navigation*, *Searching*, *Product*, *Display*, *Layout*, and other sub-categories. Yet another organization principle suggested by van Welie and van der Veer is to categorize patterns according to *user tasks* and *user type*. A user task can be selecting things, finding things and sorting. This can be done by different types of users, namely novice users, intermediate users, and expert users.

VI. Relationships

Relationships between design patterns are a key concept to gain the full reuse potential of individual design patterns. In HCI patterns, relationships are typically described very briefly, only specifying the connections to other patterns which may be applicable to a particular design problem. However, proper consideration of relationships promises

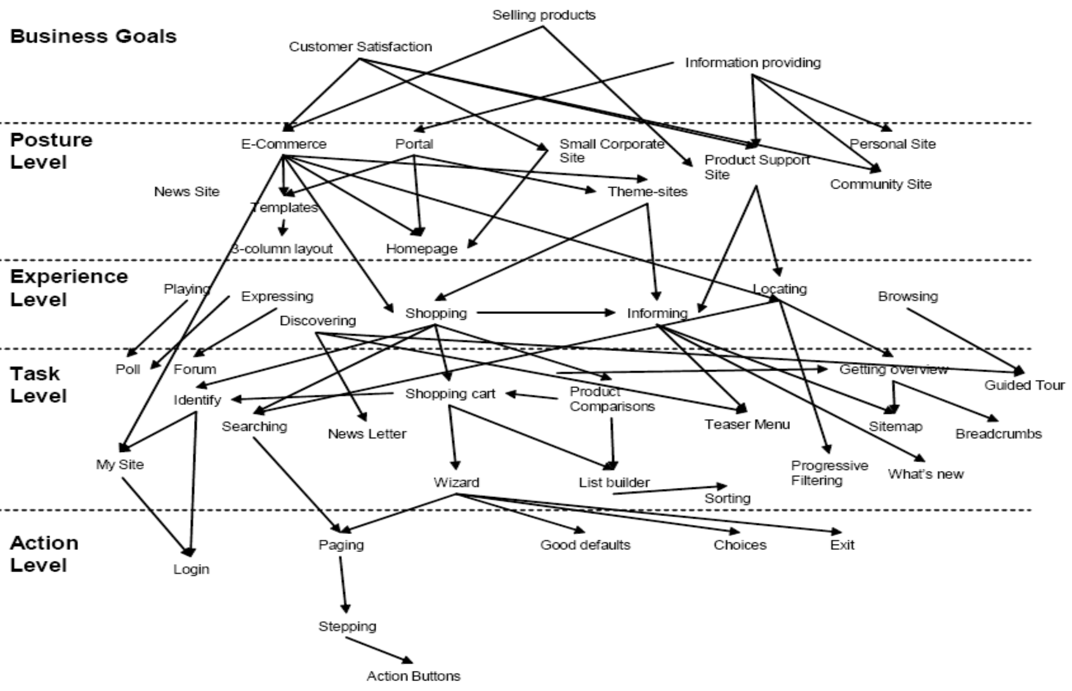


Figure 3: A part of a pattern language for Web design with focus on shopping [62]

even more powerful search and navigation opportunities. In the past, software engineering researchers have proposed possible categorization approaches for relationships in the domain of OOP patterns.

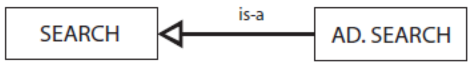


Figure 4: Specialization - The AD.SEARCH Pattern "is-a" specialization of the SEARCH PATTERN

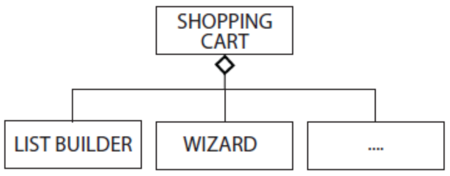


Figure 5: Aggregation - SHOPPING CARD consists of one or more design patterns.

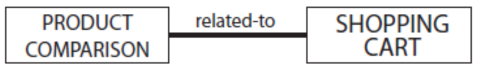


Figure 6: Association - SHOPPING CARD is "related to" PRODUCT COMPARISON

Primarily, relationships help to understand the complex interdependencies among design patterns. Pattern users can use relationship information in addition to the aforementioned pattern classifications to identify patterns which are applicable to specific design problems. Furthermore, relationships can be exploited for browsing large re-use repositories [36]. To improve and quicken the finding process, the browsing paradigm can be combined with the search paradigm as well. First the user searches for a specific design problem and based on the query results it is possible to browse through the repository to identify the best matching solution.

Noble's [50] proposal consists of three primary relationships (a pattern *uses* another pattern, a pattern *refines* another pattern, a pattern *conflicts* with another pattern) and a number of secondary relationships such as *used by*, *refined by*, *variant*, *similar*, *combines*, and others.

Zimmer's [67] approach deals with the classification of relationships in Gamma's [25] design patterns collection. He classifies relationships into three categories: a pattern *uses* another pattern in its solution, a pattern *is similar* to another pattern, and a pattern can be *combined* with another pattern. Beside this categorization, it is possible to modify existing relationships to use their altered version between different patterns. The application of categorized relationships allows to structure patterns in different layers. Zimmer has identified three semantically different layers: *basic design patterns and techniques*, *design pattern for typical software problems*, and *design patterns specific to an application domain*. Van Welie and van der Veer have indentified relationships similar to the relationship between classes in the software engineering domain [62]. They are using *Association*, *Specialization* and *Aggregation* to describe their identified relationships among design patterns. To illustrate the relationships they have provided examples how the relationships work. Figures 4, 5 and 6 show a summary of their explanations.

Beside the relationships discovered by van Welie we have investigated another one. It is called Anti-Association. It is similar to Association but it is a connection to a so-called anti-pattern, a pattern describing a bad solution approach.

VII. Semantics

Beside the relationships, semantics of design patterns can be described using ontology. Throughout the Web community there exist many design patterns which describe the same problem but with a different vocabulary. So it is difficult to understand and to access this design knowledge. Therefore, an ontology or formalized semantics are necessary to provide a common vocabulary and a machine processable form of design patterns to be used by pattern management tools.

Over the years several approaches have been developed to overcome the aforementioned problem. Below we describe some of the research activities on this topic regarding HCI design patterns.

Montero et al. describe Web design patterns using DAML+OIL [48]. In their approach they are dealing with knowledge from two different areas. On the one hand there is the *Hypermedia Models* area which describes the elements of Web applications and is defined in four basic terms:

- **Node** – a place holder which contains a number of content elements.
- **Content** – a unit of information.
- **Link** – a connection between two or more nodes or contents.
- **Anchor** – the source or target of a link.

On the other hand, the *Design Patterns* area which represents design patterns with respect to their essential content elements. Therefore, a pattern in their ontology is defined in five different terms:

- **Name** – identifies the design pattern.
- **Category** – is used for classifying the pattern.
- **Problem** – describes the context in which the pattern can be applied and the problem it addresses.
- **Solution** – shows how the problem can be solved.
- **Related Patterns** – is referencing other similar or complementary patterns.

The ontology itself is specified in DAML+OIL [57], and is subdivided into three layers. The first layer represents the pattern and hypermedia elements and is the basis for the second layer, which represents the set of hypermedia design patterns. Finally, the instances of the hypermedia design pattern layer represent the third layer. For a more detailed specification and examples see [48].

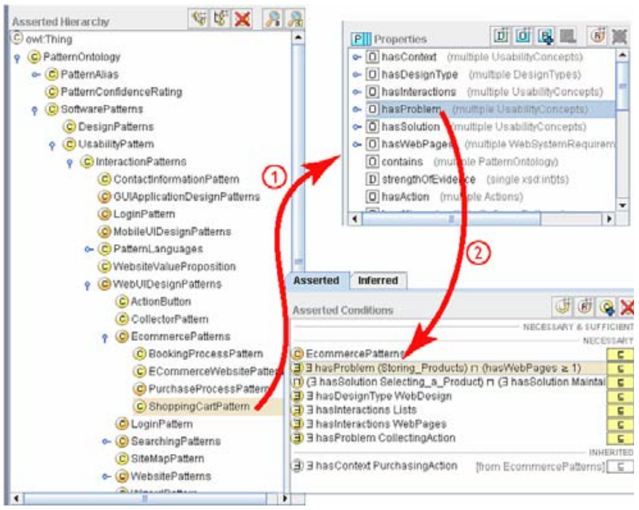
Another approach is used by Scott Henninger and colleagues [31][32][33][34][35]. They are focusing on the development of a Web-based ontology to represent design patterns which are computed by agents. Their research goal is to put the loosely coupled pattern collections into strongly coupled pattern languages which represent the context in which usability patterns can be applied. Furthermore, it was important to find mechanisms for validating design patterns. Tool support plays a crucial role in their approach because it should be possible to get useful patterns for a specific design problem when going through a question and answer (Q&A) sequence. The results of which will be computed by an inference engine.

Henninger uses OWL (Web Ontology Language) [52] to define a metamodel for intelligent pattern languages. The metamodel describes pattern properties. Some of the properties Henninger is using are [33]:

- **hasProblem** – describes the design needs of an actor for which the pattern was created.
- **hasForces** – addresses constraints and tradeoff in choosing the solution suggested in the pattern.
- **hasSolution** – tells the user which actions must be taken to solve the problem.
- **hasContext** – sets the context where the design pattern can be useful
- **hasRationale** – describes why the solution is effective.

Beside these generic properties other local semantics and range restrictions are defined in the metamodel due to the fact that the metamodel supports different types of design pattern concepts (i.e. OOP-design patterns, HCI design patterns). Furthermore, the metamodel contains several types of semantic relationships to describe the connections between design patterns:

- **uses** – Pattern A uses Pattern B if the usage is optional [67].



```
Asserted Hierarchy
owl:Thing
  PatternOntology
    PatternAlias
    PatternConfidenceRating
    SoftwarePatterns
      DesignPatterns
      UsabilityPattern
        InteractionPatterns
          ContactInformationPattern
          GUIApplicationDesignPatterns
          LoginPattern
          MobileUIDesignPatterns
          PatternLanguages
          WebsiteValueProposition
          WebUIDesignPatterns
            ActionButton
            CollectorPattern
            EcommercePatterns
              BookingProcessPattern
              ECommerceWebsitePatte
              PurchaseProcessPattern
              ShoppingCartPattern
            LoginPattern
            SearchingPatterns
            SiteMapPattern
            WebsitePatterns
```

```
Properties
hasContext        (multiple UsabilityConcepts)
hasDesignType     (multiple DesignTypes)
hasInteractions   (multiple UsabilityConcepts)
hasProblem        (multiple UsabilityConcepts)
hasSolution       (multiple UsabilityConcepts)
hasWebPages       (multiple WebSystemRequirem
contains          (multiple PatternOntology)
strengthOfEvidence  (single xsd:int|ts)
hasAction         (multiple Actions)
```

```
Asserted   Inferred
Asserted Conditions
                                             NECESSARY & SUFFICIENT
                                                          NECESSARY
EcommercePatterns
∃ hasProblem (Storing_Products) ⊓ (hasWebPages ≥ 1)
(∃ hasSolution Selecting_a_Product) ⊓ (∃ hasSolution Maintai
∃ hasDesignType WebDesign
∃ hasInteractions Lists
∃ hasInteractions WebPages
∃ hasProblem CollectingAction
                                                          INHERITED
∃ hasContext PurchasingAction        [from EcommercePatterns]
```

Figure 7: OWL Description of a Usability Design Pattern [33]

- **requires** – Pattern A requires Pattern B [67].
- **alternative** – Two patterns are alternative if they share the same problem and context but exhibit different solutions [33].
- **conflictsWith** – Pattern A conflicts with Pattern B if they should not be used together in a design [33].

Figure 7 shows a part of an instance of a usability pattern with the developed metamodel. The metamodel builds on the HCI design pattern standardization approach by Fincher et al. called PLML (cf. Section IX) and is enriched with semantically meaningful pattern descriptions and relationships between patterns. The pattern in Figure 4 is a SHOPPING CART pattern which addresses the problem of storing products that a user has selected. This is accomplished with the property restriction "hasProblem (Storing_Products ⊓ hasWebPages ≥ 1)".

The restriction is defined in OWL DL (OWL Description Logic) to formalize the properties for OWL reasoners. In addition to the property *hasProblem* the pattern has other properties which define the problem and requirements on possible solutions. For a more in-depth description of Henninger's design pattern metamodel see [33].

To facilitate tool support Henninger combines BORE [30, 32] and the semantic Web representation of design patterns. BORE (Building on Organizational Repository of Experiences) is used to demonstrate semantic Web technologies to support the design of user interfaces. The design tool can define a methodology with a set of activities which describe the development process. It is using Q&As to customize the methodology which consists of all possible activities which are necessary to design a user interface. BORE builds on the experiences of many usability projects and various contexts and it uses a rule-based representation that captures the requirements of the system.

VIII. Evaluation and Validation of Design Patterns

There are a lot of HCI design patterns available in the community. Some are good and some are less valuable. The usefulness of a pattern is often subject to the eye of the beholder. But how do we measure the usefulness of design patterns according to quality criteria or formal metrics? We have investigated two approaches which are dealing with the evaluation of patterns and pattern catalogues in HCI as well as in the software engineering domain in a structured way.

Wurhofer et al. presents a *Quality Criteria Framework* which features five main quality criteria for HCI design patterns [63] and which is based on approaches from different researchers [10][39][47][49]. Figure 8 shows a summary of the quality criteria suggested by Wurhofer. In the following we give a short summary of each criterion suggested by the framework.

- **Findability** – means that a pattern must be found easily within a pattern language / collection. This implies that they must have a meaningful categorization and pattern name.
- **Understandability** – demands that the patterns' content elements (name, problem, solution,…) must be written in an clear and simple to understand manner. This is achieved by the below described sub-criteria:
  - *Completeness of Information* – states that that the pattern must carry all relevant information (forces, problem, context, solution, example, etc.).
  - *Language* – means that the pattern must be written in easy understandable terms and short sentences.
  - *Problem-Centeredness* – all parts of the pattern should be centered on the problem and the problem solution relationship must be clear.
  - *Balance between Concreteness and Abstractness* – the pattern should not be too abstract nor to detailed.
  - *Comprehensiveness of Pattern Parts* – this criterion ensures that each part of the pattern covers everything important to the user.
- **Helpfulness** – ensures that each pattern is written in such a style that the pattern gives the user as much information as possible to implement it worries. This criterion is achieved through six sub-criteria:
  - *Improvement of Design / Architecture* – the quality of a pattern is verified if it helps to improve the design or development of a system.
  - *Problem Solving* – the pattern should help to avoid common pitfalls by using common solutions to the addressed problem.
  - *Support of Communication* – states that the design pattern should serve as the common "language" for all stakeholders.
  - *Capturing of Knowledge* – this criterion stands for the reuse aspect of the design pattern. It should capture relevant knowledge in its domain to the user.
  - *Memorability* – the main idea of a pattern must be kept in mind of the stakeholders. This can be achieved using an appropriate and easy to remember pattern name or a good sensitizing image.
  - *Feasibility* – the patterns solution should be realized easily.
- **Empirical Verification** – means that a pattern based on empirical studies has a higher quality than patterns based on personal experience.
- **Overall Acceptability** – states how much a pattern user agrees with the pattern's content. To fulfill this criterion it is important to support the pattern user's subjective acceptance of a pattern. This can be achieved by increasing the *Overall Believe in Pattern* and the *Overall Agreement with Pattern*.

With this criteria catalogue it is possible to validate design patterns according to their quality.

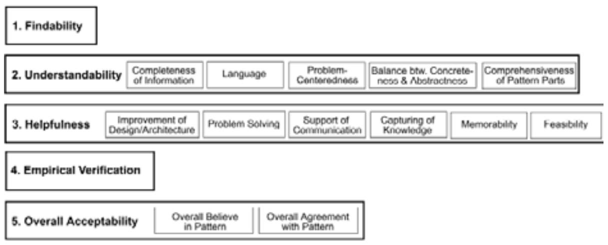

1. Findability
2. Understandability
Completeness of Information
Language
Problem-Centeredness
Balance btw. Concrete-ness & Abstractness
Comprehensiveness of Pattern Parts
3. Helpfulness
Improvement of Design/Architecture
Problem Solving
Support of Communication
Capturing of Knowledge
Memorability
Feasibility
4. Empirical Verification
5. Overall Acceptability
Overall Believe in Pattern
Overall Agreement with Pattern

Figure 8: Components of the Quality Criteria Framework [63]

Another approach has been developed by Cutumisu et al. [16]. They propose how to evaluate the effectiveness of a design pattern catalogue or compare different catalogues according to their effectiveness. The authors developed a metric with which it is possible to validate patterns within an existing design pattern catalogue. They defined four metrics which are dependent on a specific application. That means the metrics take the patterns which are used in a specific application and compare them using various formulae to the used pattern catalogue. Cutumisu et al. define the four metrics as follows:

- ***usage** – is the ratio of patterns used in the application that come from the catalogue to the total number of patterns in the catalogue.*
- ***coverage** – is the ratio of catalogue patterns used in the application to the total number of patterns used in the application.*
- ***utility** – is the ratio of pattern instances in the application whose patterns are in the catalogue to the total number of patterns used in the application that come from the catalogues.*
- ***precision** – is the ratio of the total number of patterns used in the application that come from the catalogue to the number of adaptations required for these pattern instances.*

If a pattern catalogue has a high *usage*, *coverage*, *utility*, and *precision* it is, according to Cutumisu et al. a good pattern catalogue. Although the pattern metrics are designed for software patterns, it is easy to adapt them to the HCI design pattern domain. For a more detailed description and the equations for each of the metrics see [16].

IX. TOOL SUPPORT

There are various tools which are exploiting the reuse potential of HCI design patterns. These tools can be categorized into online libraries / catalogues, pattern management tools, and pattern-based UI design tools. Due to space limitation we describe, in our mind, the most important ones.

A. Pattern Libraries / Collections

Pattern libraries or collections are focusing on the categorization and dissemination of patterns via the Internet. Sometimes there are basic mechanisms provided to create and submit patterns to a repository. Such an online pattern library is the *Yahoo! Design Pattern Library* [43, 66], a part of the Yahoo! Developer Network. The founder's intention of the Yahoo! Library was to provide a tool to increase the consistency and usability across Yahoo! and the productivity of the UI design team. Today the design pattern library is an often-used tool for UI designers and researchers. Currently, there are about 47 patterns in six categories available. Each pattern undergoes an extensive review process within Yahoo!. They are reviewed, revised, and rated. After review, the patterns are published and made available to the public. All patterns in this library are under the Creative Commons Attribution 2.5 License (June 2009). Main features of the library are:

- A *blogging tool* for discussing patterns in the library.
- A *history function* which helps users to see which changes were made over time.

Further online pattern libraries are Welie.com [64], which provides 130 UI design patterns with the possibility to export them to PLML [22]. Furthermore, website users can comment and discuss certain patterns.

As an addition to her *Designing Interfaces* book [60], Tidwell provides a pattern library with selected patterns where she updates and publishes new patterns. This library is available at [60].

B. Pattern Management Tools

Pattern management tools are focusing on manipulating patterns, navigating through pattern libraries, and providing mechanisms to add relationships between patterns to create a pattern language. They are easy to access and pattern users can communicate with others via the pattern repository.

MOUDIL (Montreal Online Usability Patterns Digital Library) [26] is a comprehensive framework for capturing and disseminating patterns. It provides features and tools like:

- Submission of patterns in *different formats*.
- International *review and validation* of submitted patterns.
- A *pattern editor* for adding semantic information to the patterns.
- A *pattern navigator* which allows navigating in different ways through the pattern library.
- A *pattern viewer* which provides different views of the pattern.

Unfortunately, the prototype of this pattern library is not longer available online.

Currently under development is another online pattern management tool which employs XPLML [40], an improved version of PLML. XPLML provides a set of common content elements, and it is possible to add semantic information to design patterns. The tool will offer features such as:

- A *pattern editor* with functions to support pattern authors in writing and updating design patterns.
- A *design pattern language visualization tool* for presenting relationships between patterns in a pattern language.
- The *pattern form transformation* allows pattern users to change the presentation form of a design pattern. For example, if a user prefers the Alexandrian form, the tool provides mechanisms to change the pattern form from e.g. Tidwell's to the preferred (Alexandrian) form in order to maximize user acceptance.
- A *wiki functionality* which should involve all interested users in developing new and improving existing patterns.

C. Pattern-based UI design tools

The last category describes pattern-based UI design tools. They provide functions for using design patterns in UI design activities. Patterns are used for generating user interfaces in a semi-automatic way. These tools usually provide a defined set of UI patterns, which can be used within the tool as building blocks to create the UI system.

PIM (Patterns in Modeling) [54], a model-based UI development tool, aims to support UI designers in composing the UI models through pattern application. With *PIM* it is possible to develop user interfaces on a more abstract and conceptual way. This helps designers to handle very complex systems more easily. Users can put their attention on conceptual properties rather than being distracted by technical and implementation details.

A further tool, developed by Ahmed and Ashraf, is called *Task Pattern Wizard* [2]. It is based on XUL (XML User Interface Language) [65] to describe the patterns and models. UI design patterns are used as modules for establishing task, dialog, presentation, and layout models. The tool guides the UI designer through the pattern adaption and integration process and it provides functions for using, selecting, adapting, and applying patterns within the proposed framework PD-MBUI (Pattern-Driven and Model-Based User Interface). The framework tries to unify the pattern-driven and model-based approaches, two methods for UI and software engineering. A more detailed description of the framework is given in [2].

DAMASK, developed by Lin and Landay [44], is a prototyping tool to produce Web UI's across different devices with the support of design patterns. The tool relies on two components. The *layer component* specifies which parts of the UI can be used across all devices and which can only be used on a single device. The second component is the *pattern component*: In *DAMASK*, an HCI design pattern consists of pre-defined UI elements that are optimized for each device. The pattern repository of *DAMASK* has 90 patterns from "The Design of Sites" [18] which can be extended by the UI designer. The UI designer sketches out a UI for one device and *DAMASK* constructs an abstract model from which it generates the UI's for the other devices. Once the first layout is established, the UI designer can refine this layout and *DAMASK* changes the UI's for the other devices accordingly. Furthermore, the tool provides a function for testing the established UI's.

X. STANDARDIZATION APPROACHES

To our knowledge, the only serious standardization approach was started at a workshop at a Human-Computer Interaction conference in 2003. Due to the vast amount of design pattern forms, Fincher et al. [22] proposed a standard pattern form for HCI patterns called PLML (pronounced "pell mell"). The goal was to provide a standard pattern form where common elements should help pattern authors and users to use design patterns across different collections. PLML is specified in XML and comprises 16 content elements on which the workshop participants agreed. It turns out that only van Welie's pattern collection [64] makes use of PLML. He provides an export function to transform patterns from his collection to PLML. However, this approach suffers from certain technical limitations as Kamthan points out [38]. He mentions that the design principles behind the PLML DTD are not specified and that elements are not strictly enough defined, because of the broad use of the XML ANY element in the specification. Kamthan also points out that PLML does not describe semantic relationships between patterns, which are necessary when using PLML in a pattern language.

Since the publication of PLML, researchers tried to improve it. PLML v. 1.2. developed by Deng et al. [17], is an augmented PLML with some additional elements but does not solve serious shortcomings such as the lack of formalized relationships among patterns.

XI. CONLUSION AND FUTURE WORK

The concept of HCI design patterns is widely accepted tool to represent design knowledge in a reusable format. In the last years many concepts concerning the components of HCI patterns were proposed, such as pattern forms, organizing principles, standardization approaches, ontology, evaluation and verification of patterns. This diversity leads to blurred conceptualization and may confuse especially novice users. To exploit the full reuse potential of patterns, a unification of the above discussed components should be established, which does not constrain pattern authors in their work but supports pattern users by easing understanding and instantiation of patterns to specific design problems [40]. Therefore a universal pattern form needs to be established and enriched with semantics. This article shows that there are many HCI design pattern resources available on the WWW and because of the vast amount of different design patterns available there exists many different forms as well. To overcome the problem, an appropriate ontology would help to share and disseminate HCI design patterns among different repositories and help the authors and pattern users to work more efficient with the provided design knowledge. Therefore, a lot of research must be undertaken which includes analyzing the different design pattern forms and examining the content elements' semantics. Furthermore, to agree on a "standard" pattern form it is necessary to discuss the results of the above mention research with the HCI design pattern community to agree on a unified HCI design pattern structure.

REFERENCES

[1] M. Adams, J. Coplien, R. Gamoke, R. Hanmer, F. Keeve, and K. Nicodemus, "Fault-Tolerant Telecommunications System Patterns", in Pattern Languages of Program Design 2, pp. 549-562, Addison-Wesley, 1996

[2] S. Ahmed and G. Ashraf, "Model-based User Interface Engineering with Design Patterns", in Journal of Systems and Software, vol. 80, pp. 1408-1422, 2007

[3] C. Alexander, S. Ishikawa, and M. Silverstein, "A Pattern Language", Oxford University Press, 1977

[4] C. Alexander, "Notes on the Synthesis of Form", Harvard University Press, 1964

[5] C. Alexander, "The Oregon Experiment", Oxford University Press, 1975

[6] C. Alexander, "The Timeless Way of Building", Oxford University Press, 1979

[7] E. Bayle, "Putting it all together: Towards a Pattern Language for Interaction Design: A CHI workshop", in SIGCHI Bulletin, vol. 30, pp. 17 – 23, 1998

[8] K. Beck and W. Cunningham, "Using Pattern Languages for Object-Oriented Programs", OOPSLA'87:Workshop on the Specification and Design for Object-Oriented Programming, 1987

[9] J. Borchers, "CHI Meets PLoP: An Interaction Patterns Workshop", SIGCHI Bulletin, vol. 32, 2000

[10] J. Borchers and J. Thomas, "Patterns: What's in it for HCI?", CHI'01: Extended Abstracts on Human Factors in Computing Systems, ACM Press, pp. 225 - 226, 2001

[11] J. Borchers, "A Pattern Approach to Interactive Design", Wiley, 2001

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, „Pattern-Oriented Software Architecture: A System of Patterns", vol. 1, Wiley, 1996

[13] E. Chung, J. Hong, J. Lin, M. Prabaker, J. Landay, and A. Liu, "Development and Evaluation of Emerging Design Patterns for Ubiquitous Copmuting", in Proc. Of Designing Interactive Systems, 2004

[14] T. Coram and J. Lee, "Experiences – A Pattern Language for User Interface Design, 1996, Available at: http://www.maplefish.com/todd/papers/Experiences.html, Accessed on: June 30, 2010

[15] C. Crumlish and E. Malone, "Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience", O'Reilly Media, 2009

[16] M. Cutumisu, C. Onuczko, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, J. Siegel, and M. Carbonaro, "Evaluating Pattern Catalogs: The Computer Games Experience", ICSE '06: Proceedings of the 28th International Conference on Software Engineering, ACM, pp. 132—141, 2006

[17] J. Deng, E. Kemp, and G. Todd, "Focussing on a Standard Pattern Form: The Demelopment and Evaluation of MUIP", in Proc. of the Seventh ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction, pp. 83-90, ACM Press, 2006

[18] D. van Duyne, J. Landay, and J. Hong, "The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience", Addison-Wesley, 2003

[19] T. Erickson, "The Interaction Design Patterns Page", Available at: http://www.visi.com/~snowfall/InteractionPatterns.html, Accessed on June 30, 2010

[20] S. Fincher, "The Pattern Gallery", Available at: http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html, Accessed on June 30, 2010

[21] S. Fincher, J. Finlay, S. Greene, L. Jones, P. Matchen, J. Thomas, and P. Molina, "Perspectives on HCI Patterns: Concept and Tools", CHI'03: Extended Abstracts on Human Factors in Computing Systems, ACM Press, pp. 1044 – 1045, 2003

[22] S. Fincher, "Perspectives on HCI Patterns: Concepts and Tools (introducing PLML)", Interfaces, vol. 56, pp.26-28, 2003

[23] S. Fincher, "The Pattern Gallery", Available at: http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html, Accessed on June 30, 2010

[24] S. Fincher and P. Windsor, "Why Patterns are not enough: Some Suggestions Concerning an Organising Principle for Patterns of UI Design", CHI'2000 Workshop on Pattern Languages for Interaction Design: Building Momentum, 2000

[25] E. Gamma, R. Helm, R. Johnson, andJ. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Reading, 1994

[26] A. Gaffar, H. Javahery, A. Seffah, and D. Sinning, "MOUDIL: A Comprehensive Framework for Disseminating and Sharing HCI Patterns", CHI'03 workshop: Perspectives on HCI patterns: Concepts and Tools, 2003

[27] I. Graham, "A Pattern Language of Web Usability", Addison-Wesley, 2003

[28] R. Griffiths, L. Pemberton, and J. Borchers, "Usability Pattern Language Workshop", at INTERACT'99, Available at: http://www.it.bton.ac.uk/staff/rng/UPLworkshop99/PositionPapers.html, Accessed on: June 30, 2010

[29] "HCIPATTERNS.ORG", Available at: http://www.hcipatterns.org/, Accessed on June 30, 2010

[30] S. Henninger, "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle", ICSR '96: Proceedings of the 4th International Conference on Software Reuse, IEEE Computer Society, 1996

[31] S. Henninger, M. Keshk, and R Kinworthy, "Capturing and Disseminating Usability Patterns with Semantic Web technology", Workshop at CHI 2003: Perspectives on HCI Patterns: Concepts and Tools, 2003

[32] S. Henninger, "Tool Support for Experience-Based Software Development Methodologies", in Advances in Computing, vol. 59, pp. 29 – 82, 2003

[33] S. Henninger and P. Ashokkumar, "An Ontology-Based Metamodel for Software Patterns", 18th International Conference on Software Engineering and Knowledge Engineering (SEKE2006), pp. 327 – 330, 2006

[34] S. Henninger, "Disseminating Usability Design Knowledge through Ontology-Based Pattern Languages", Proc. Semantic Web User Interaction Workshop (ISWC2006). 2006

[35] S.Henninger and P. Ashokkumar, "An Ontology-Based Infrastructure for Usability Design Patterns", Proc. First International Workshop Semantic Web Enabled Software Engineering, 2005

[36] M. Hitz and H. Werthner, A Graph Oriented Approach to Enhance Reusability in *-bases, WISR'92: 5th Annual Workshop on Software Reusability, 1992

[37] M. Irons, "Patterns for Personal Web Sites", Available at: http://www.rdrop.com/~half/Creations/Writings/Web.patterns/, Accessed on: June 30, 2010

[38] P. Kamthan, "A Critique of Pattern Language Markup Language", Interfaces, vol. 68, pp. 14-15, 2006

[39] D. Khazanchi, J. Murphy, and S. Petter, "Guidelines for evaluating patterns in the IS domain", MWAIS 2008 Proceedigs, 2008, Paper 24, Available at : http://aisel.aisnet.org/mwais2008/24, Accessed on: June 30, 2010

[40] C. Kruschitz, "XPLML: A HCI Pattern Formalizing and Unifying Approach", in Proc. of the 27th International Conference Extended Abstracts on Human Factors in Computing Systems, pp. 4117 – 4122, ACM, 2009

[41] C. Kruschitz and M. Hitz, "The Anatomy of HCI Design Patterns", Proc. of Computation World; Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, IEEE Computer Society, pp. 202 – 207, 2009

[42] T. Kunert, "User-Centered Interaction Design Patterns for Interactive Digital Television Applications", Springer, 2009

[43] M. Leacock, E. Malone, and C. Wheeler, "Implementing a Pattern Library in the Real World: A Yahoo! Case Study", ASIS&T IA Summit, 2005

[44] J. Lin and J. Landay, "Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces", in Proc. 26th International Conference on Human Factors in Computing Systems, pp. 1313-1322, ACM, 2008

[45] "Littles Springs Design – Mobile UI Design Resources", Available at: http://patterns.littlespringsdesign.com/index.php/Main_Page, Accessed on: January 18, 2010

[46] M. Mahemoff and L. Johnston, "Pattern Language for Usability: An Investigation of Alternative Approaches", in Proc. Third Asian-Pacific Conference in Computer Human Interaction, pp. 25-31, IEEE Coomputer Society, 1998

[47] K. McGee, "Patterns and Computer Game Design Innovation", IE'07: Proc. of the 4th Australasian Conference on Interactive Entertainment, pp. 1 – 8, 2007

[48] S. Montero, P. Díaz, and I. Aedo, "Formalization of Web Design Patterns using Ontologies", Proc. Of the 1st International Atlantic Web Intelligence Conference, Springer-Verlag,Vol. 2663, pp. 179 – 188, 2003

[49] S. Nieburg, K. Kohler, and C. Graf, "Engaging Patterns: Challenges and Means Shown by an Example", Engineering Interactive Systems, Springer, pp. 586 – 600, 2008

[50] J. Noble, "Classifying Relationships Between Object-Oriented Design Patterns", in Proc. of the Australian Software Engineering Conference, 1998, IEEE Computer Society

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

[52] "OWL – Web Ontology Language", Available at: http://www.w3.org/TR/2004/REC-owl-guide-20040210/, Accessed on June 30, 2010

[53] "Portland Pattern Repository", Available at: http://c2.com/ppr/, Accessed on January 5, 2010

[54] F. Radeke, P. Fobrig, A. Seffah, and D. Sining, „PIM Tool: Support for Pattern-Driven and Model-Based UI Development", 5th International Workshop: Task Models and Diagrams for User Interface Design, LNCS: Programming and Software Engineering, vol 4385, pp. 82-96, 2007

[55] T. Schuemmer and S. Lukosch, "Patterns for Computer-Mediated Interaction", John Wiley & Son, 2007

[56] R. Smith, "Panel in Design Methodology", OOPSLA'87:Addendum to the Proceedings on Object-Oriented Programming Systems, Languages and Applications, pp. 91-95, ACM, 1987

[57] "The DARPA Agent Markup Language", Available at: http://www.daml.org, Accessed on June 30, 2010

[58] "The Hillside Group", Available at: http://hillside.net, Accessed on June 30, 2010

[59] J. Tidwell, "Common Ground", Available at: http://www.mit.edu/~jtidwell/common_ground.html, Accessed on June 30, 2010

[60] J. Tidwell, "Designing Interfaces", O'Reilly, 2005 Available at: http://www.designinginterfaces.com, Accessed on June 30, 2010

[61] "UI Patterns - User Interface Design Pattern Library", Available at: http://ui-patterns.com, Accessed on June 30, 2010

[62] M. van Welie and G. van der Veer, "Pattern Languages in Interaction Design: Structure and Organization", Human-Computer Interaction – INTERACT'03, pp.527-534, IOS Press, 2003

[63] D. Wurhofer, M. Obrist, E. Beck, and M. Tscheligi, “Introducing a Comprehensive Quality Criteria Framework for Validating Patterns”, Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World, pp. 242 – 247, 2009

[64] “Welie.com – Patterns in Interaction Design”, Available at: http://www.welie.com, Accessed on June 30, 2010

[65] “XML User Interface Language”, Available at: https://developer.mozilla.org/En/XUL, Accessed on June 30, 2010

[66] “Yahoo! Design Pattern Library”, Available at: http://developer.yahoo.com/ypatterns/, Accessed on June 30, 2010

[67] W. Zimmer, “Relationships between Design Patterns”, Pattern Language of Program Design, Addison-Wesley, 1994