

# Disseminating Usability Design Knowledge through Ontology-Based Pattern Languages

Scott Henninger

Univ. of Nebraska-Lincoln, Computer Science and Eng., CC 0112,  
Lincoln, NE 68588-0112 USA  
scotth@cse.unl.edu

**Abstract.** Usability patterns represent knowledge about known ways to design graphical user interfaces that are usable and meet the needs and expectations of users. There is currently a plethora of usability patterns published in books, private repositories and the World-Wide Web. The dominance of pattern discovery efforts has neglected the emerging need to organize the patterns so they can become a proactive resource for developing interfaces. This paper presents an approach using Semantic Web concepts that turns informal patterns into formal representations capable of supporting systematic design methods. Preliminary research is described that aims to turn loosely coupled pattern collections into strongly coupled pattern languages that help organize usability knowledge into a form that is easily and widely disseminated. This in turn can be used to facilitate the accumulation of usability development knowledge.

## 1 Usability Patterns as an Interface Design Resource

The development of interactive software systems, i.e. systems with significant user interface components, is currently faced with a dilemma. Design for usability is becoming increasingly important to the success of software systems, but software developers are usually poorly trained in human factors and usability issues. Education, use of HCI specialists in the development process, and iterative development processes aimed at evaluating and improving the user interfaces [18] are necessary and cannot be fully replaced. But the abundance of error-prone and poorly designed user interfaces that exist in modern software systems indicates that complementary techniques are needed to disseminate usability design knowledge and best practices early in the design and development process.

There is no lack of information and guidance on the design, development, and evaluation of user interfaces. Usability guidelines, patterns, principles, books, Web pages depicting good and bad examples, databases and various repositories are examples of both the plethora of knowledge and proliferation of formats that have been used to disseminate usability design knowledge. But current approaches to representing usability knowledge are document-based, at best supported with hypertext tools and/or Web page catalogs. These passive representations rely on individual developers to know of the existence of relevant knowledge sources, extract useful information, and understand how and when the obtained resources should be applied. As the size of the body of knowledge continues to grow in the current fractured manner, this

method becomes, or has already become, untenable. Tools and techniques are needed that create an interconnected corpus of knowledge with a degree of agreement within the community and that can be refined and evolved to meet the ever changing demands of business and technology domains.

The main objectives of this research are to: 1) construct formal computational frameworks for creating interconnected corpora of usability knowledge; and 2) build tools that put context-appropriate usability guidelines at the fingertips of software designers and usability specialists so they can be used early and throughout the design and development process.

In this paper a framework is presented in which usability patterns [7, 16] are used for representing usability knowledge. Semantic Web ontologies [23] are used as a distributed medium for formally defining pattern attributes and relationships between the patterns using Web-based standards. The ontologies are used to organize loosely coupled pattern collections into pattern languages capable of systematic usability design support. The choice of pattern formats and Semantic Web technologies are chosen purposefully for their ability to federate distributed heterogeneous information over the WWW and to facilitate a degree of standardization within the community. This facilitates the development of an interconnected corpus of knowledge that embodies a degree of consensus within the design community.

In the following sections, usability patterns and the types of tools and support currently available for applying usability and other software development guidelines and patterns are described. Some general background is given on using Semantic Web technologies to implement pattern languages, followed by an example of ontologies and associated rules and inferences for intelligent usability pattern support.

## **2 Usability Guidelines and Patterns**

Usability guidelines have been used as a means to disseminate usability knowledge and ensure a degree of consistency across applications [21, 29]. While hundreds of usability guidelines have been designed and published, empirical studies have shown mixed results, with some demonstrating that both novice and expert HCI specialists benefit from guidelines [10, 22], while others revealing challenges with finding and applying guidelines for specific problem settings [32, 33]. However, all have found significant problems with the manner in which the knowledge is disseminated and applied. These and other problems stem from the abstract and decontextualized nature of current guideline techniques [12, 20], creating a mismatch with the cognitive state of developers, who tend to “ask questions about specific problems they have with their own design rather than abstract ones” [3].

A usability patterns community, [7, 19, 37, 39] inspired by work on software design patterns [17], has begun to explore how patterns can be used to provide enhanced representation for usability knowledge that explicitly defines context and interrelationships between patterns [15]. The essential idea of a design pattern is to capture successful solutions to recurring problems with the context and forces that operate on the problem to yield a general, repeatable, solution [2].

Differences between usability guidelines and usability patterns lie primarily in perspective and representation of the information. The perspective of usability patterns tends to be more problem-oriented, focusing on describing a problem and solution, rather than the more general information or advice perspective of guidelines. As shown by the Breadcrumb pattern [38] in Figure 1, patterns also add standard property fields to explicitly describe the context of the problem and the forces that shape the problem and its variants. Yet the basic goals of these approaches are essentially the same: to document and manage collective knowledge about usability design issues in a format that is easily disseminated and understood.



Figure 1: The Breadcrumb Pattern.

## 2.1 Patterns, Pattern Collections and Pattern Languages

A software pattern is a formatted or free-text description of abstract problems that recur in a certain context coupled with a validated solution that can be contextualized (or parameterized) by forces [2, 17]. Figure 1 displays one of many pattern formats [13] that have attributes for a name, problem, context, solution, rationale and example. The majority of existing patterns are organized in collections, or catalogs [26], of loosely coupled sets of pattern descriptions classified by defined criteria [17, 21] or a taxonomy [38]. These collections tend to be self-contained "islands" of knowledge that rarely contain pointers outside of their boundaries. Organizing the patterns into a network of higher-level patterns that are resolved or refined by more detailed patterns, a *Pattern Language* [2, 39], has not been a focus of these collections.

While the original pattern work by Christopher Alexander for Architectural design defined pattern languages as generating holistic design solutions [2], this perspective has largely been lost when applied to software patterns [1]. When discussing collec-

tions of patterns, current literature either provides a murky definition of pattern collections or uses the concept interchangeably with pattern languages.

Our research makes a clear distinction between pattern *collections* and pattern *languages* along two dimensions [26]; 1) While pattern collections are relatively isolated, pattern languages are highly interconnected [36] that focus as much on the meanings of the relationships between patterns as the patterns themselves. This leads to more robust knowledge structures with a higher probability of finding a set of patterns to work together to form the basis of a design solution; 2) In addition to relationships between patterns, pattern languages provide structuring principles that enables the generation of design solutions, whether complete or partial. For example, levels of decomposition or abstraction can be used to approach a problem top-down, from general concepts to specifics. Other examples include temporal decision sequences [9], levels of scale (architecture), and other forms that resolve design processes in an orderly fashion through systematic design and reuse of patterns. Note this is a very weak interpretation of language, as we are not imposing a formal syntax or grammar, but is well-aligned with the traditional interpretation of a “pattern language” [2].

## 2.2 Current Support for Usability Patterns

The current state of affairs for pattern users is to use collections of patterns made available through a handful of portals [8, 14, 38, 40], Wiki pages [11, 27], and books. Given the potentially copious numbers of patterns<sup>1</sup> that can be used in different contexts, and the lack of training in usability issues, this is not a satisfactory solution. Computational pattern representations are needed that facilitate context sensitive retrieval and applications that can effectively support design processes [13].

Suppose a project team is developing an E-commerce website to serve users who want to purchase a set of products through a Web browser. The product offerings are large and diverse enough that it makes sense to divide the site into multiple Web pages with navigational aids to go between categories. Some members of the team are aware that proven usability knowledge for these types of interfaces is available in the Interaction Design Patterns website (often referred to as the Amsterdam Patterns Collection) [38] containing over 60 usability patterns, including guidelines relevant to the project such as Ecommerce and Web shopping patterns.

Given the discovery of this pattern collection, the team must read, digest and organize the collection of patterns to find which ones might be applicable to parts of their interface design. This leads to a number of problems when trying to design the system using the pattern collection. First, since the patterns are not represented in a problem-oriented form, it is not immediately clear which set of patterns apply to a particular problem. For example, the “Shopping Cart” pattern [38] is a solution to the problem of users selecting items displayed in multiple Web pages that cannot be simultaneously displayed, but it is not clear which other patterns are needed in con-

---

<sup>1</sup> A recent investigation of pattern collections found more than 250 separate usability patterns [25], yet the authors missed a number of sources we happen to be aware of, such as the van Welie Amsterdam usability pattern collection [38].

junction with this one to satisfy other requirements such as purchasing items, search, comparison, and etc. The developers must read all the patterns and make decisions about the applicability of each pattern to the current project.

Second, after a particular pattern has been chosen, there are no indications or formal relationships about which pattern(s) are compatible, incompatible, or should be used with the chosen pattern. For example, using the Shopping Cart pattern may involve choices for specific interaction types, such as using a persistent button or frame to indicate items in the shopping cart, or requires certain types of search interactions. Little to no information is provided, nor are mechanisms in place, *to describe how the patterns may work together for solutions to larger problems*. In addition, if there are related patterns in other collections, such as the UI Patterns and Techniques site [34] or Yahoo! Design Pattern Library [40], there are no links to the individual patterns of interest. At most one will find a link to the entire pattern collection and pattern users will have to “sort it out” for themselves to piece together a solution.

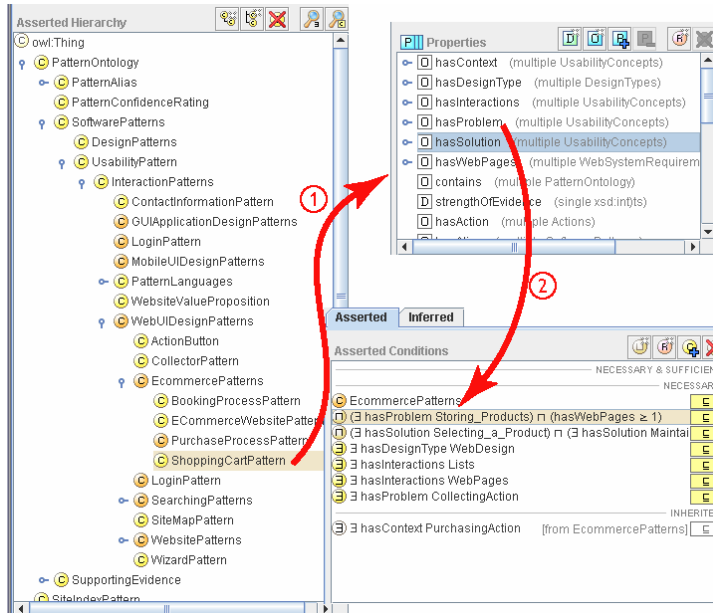
Third, if the patterns do not fully meet the needs of the development context, there is no mechanism by which the developers can extend the patterns to meet their needs. A flexible framework is needed for building pattern standards in a disciplined fashion.

### 3 Semantic Web Technologies and Pattern Languages

A major weakness of current pattern tool representations is the lack of semantic, or typed, relationships between patterns. In particular, the potential utility of using the structured format of patterns, for example using a ‘context’ field to formally or systematically indicate when a pattern should be used, has yet to be explored in any detail. The following sections describe how ontological descriptions using Semantic Web technologies can be utilized to provide typed relationships between patterns.

Semantic Web technologies are gaining widespread acceptance as a Web-based knowledge delivery technology [24]. It supports formal descriptions of information in a computational format for machine processing that also has human-readable forms. In addition, Semantic Web resources are stored on the World-Wide Web, raising possibilities for both tying multiple distributed pattern collections together while providing a computational medium that allows agents to make intelligent inferences across a distributed network of Semantic Web resources. While other formal mediums, such as UML, can be used to model and represent patterns, none have the combination of both formal representation and distributed Web accessibility afforded by Semantic Web technologies. In terms of implementing pattern languages, Semantic Web technologies provide a computational medium that can:

- intelligently match system design contexts and requirements to pattern language elements (patterns and pattern relationships)
- make intelligent inferences about applying patterns to solve problems at successive levels of abstraction, thus providing the basis for a pattern language
- automatically and dynamically classify patterns into pattern languages that can generate complete design solutions



**Figure 2:** An OWL description of a usability design pattern.

- check the consistency of patterns and pattern language attributes

After defining some of the key concepts used in this approach, some examples of potential relationships and inferences between problem descriptions (requirements) and solutions (pattern language instantiations) are described. These examples work *toward* developing tools that enable the creation of pattern languages by matching sets of patterns to usability requirements using Semantic Web ontologies.

### 3.1 Ontology-Based Pattern Representations

The definition often used for *ontology* is a “Formal, explicit specification of a shared conceptualization” [31]. An ontology consists of a set of definitions from a formal vocabulary defining a “schema” and instances (referred to as individuals) of the schema concepts. Ontologies are therefore a natural extension to the essential pattern concept of providing a common vocabulary to communicate design concepts. In a computational context, an ontology is a formal, machine readable, shared vocabulary consisting of concepts, relationships, and axiomatic definitions that can be used by standard Reasoners [4] to classify and infer new facts. Figure 2 contains a screen image of parts of an ontology developed in an ontology editor, Protégé [30], using the Web Ontology Language (OWL).

In the taxonomy (the “Asserted Hierarchy” in Figure 2), the concept ‘Usability Pattern’ is defined as a type of ‘Software Pattern’. Note that terms in this ontology

have a class/subclass relationship that has been “Asserted”, i.e. defined by an ontology designer. The classes are defined by a set of properties shown in Figure 2 (a partial list of which is pointed to by arrow ①) that are used to represent relationships between concepts. For example, ‘hasSolution’ is defined as a relationship between the concepts ‘ShoppingCartPattern’ and ‘UsabilityConcepts’. In informal terms, this means that instances of ‘ShoppingCartPattern’ are allowed to take on values from ‘UsabilityConcepts’. For example, a shopping cart design might have a specific solution involving browser frames and an icon that is always displayed.

The properties defined in Figure 2 are inherited from the ‘InteractionPatterns’ concept (see Asserted Hierarchy in Figure 2), which represents an extended version of the Pattern Language Markup Language (PLML) developed for HCI patterns [16] to include additional properties for reasoning, strength of evidence for the pattern and multiple types of “relatedTo” properties that can be computationally associated with semantic definitions.

Properties can be further refined through logical restrictions. For example, the ‘hasProblem’ property defines restrictions designed to convey the meaning that a ShoppingCartPattern is a solution to the problem of storing products from multiple web pages that a user has chosen to buy. This is represented in OWL using the restriction  $(\exists \text{ hasProblem Storing\_Products}) \sqcap (\text{hasWebPages} > 1)$  (see arrow ②). Formally this means that the Shopping Cart pattern has at least one value for the hasProblem attribute from the Storing\_Products concept, as defined by the existential quantifier ( $\exists$ ). This is joined with a logical AND (intersection in OWL) with the statement that there must be more than one hasWebPages definition.

Describing pattern attributes in such a formal manner supports inference as a complement to defined, or “asserted”, relationships. For example, any pattern that has the same problem as the Shopping Cart pattern can be inferred to be an alternative pattern as long as the contexts are same and the solutions are different. Further, if these restrictions are stated as equivalence restrictions (Necessary and Sufficient conditions) a Reasoner will infer classification, i.e. given an instance with the defined property types, the instance is inferred as a member of the defined class. Although not shown here, if the restriction “ $\exists \text{ hasDesignType WebDesign}$ ” were stated as the only equivalence restriction for membership in the ShoppingCartPattern concept, then all new patterns specifying one or more relationships of type WebDesign would be inferred to be a Shopping Cart Pattern. More complex restrictions can be used to precisely define class membership as deeply as deemed necessary by ontology designers.

### 3.2 Formal Representation of Pattern Languages

Given formal description of patterns, it is now possible to define how these patterns are combined to create a pattern *language*. By our definition, a software pattern language consists of a collection of patterns with structuring principles that support the creation of design solutions for a specific domain of software systems. For the purposes of a proof-of-concept exemplar, we have defined a specific pattern language based on van Welie’s levels of decomposition for website usability [39] that defines successive levels of problem decomposition: Posture Level, Experience Level, Task

Level and Action Level (which corresponds to widget selection). Posture level patterns describe the overall purpose of the website. They determine the site structure and the main experiences a site offers. For example, an ecommerce Website Pattern is a posture level pattern. It describes the common elements that are part of an ecommerce website. Experience level patterns describe experiences that users go through to achieve their goals described in the Posture level patterns. Typical experiences are Shopping, Locating etc. Task Level patterns such as *ShoppingCart* and *ProductComparison* perform tasks such as choosing products to buy or comparing products. They describe a series of interactions on one or more objects for solving a problem. Finally, Action level patterns describe common widgets that are used in accomplishing the various tasks described in Task level patterns.

Each of these levels defines critical information that is necessary to derive complete designs for specific problems. Translating this to a formal medium can be used to ensure that patterns for each of the levels are chosen and that the patterns chosen through all levels are consistent, i.e do not have any contradictory design criteria. For example, the Task Level, which is a level of abstraction above the Action Level (or widget level) can be formally described as patterns that have “a series of actions on one or more objects for solving a problem.” [39]. This can be formally stated as:

$$(\exists \text{hasSeries Actions}) \sqcap (\forall \text{hasSeries Actions}) \sqcap (\exists \text{hasObject UsabilityWidget}) \\ \sqcap (\text{hasObject} \geq 1).$$

Informally, this states that a task level pattern is any pattern in which all series of Actions are of type ‘Actions’ and that at least one series of Actions is defined. In addition, the pattern must have at least one association (hasObject) with a type of object, one of which has to be of type UsabilityWidget, but can also be of other, unspecified types. Comparing this to the original definition of the Task Level, we see that we have formally restricted class membership so that only patterns with a series of actions on one or more objects (with an “extra” restriction that one must be of type UsabilityWidget) can be members of the Task Level. The above restriction is described as a class equivalence condition so that pattern instances satisfying these conditions will be classified as a task level pattern by an OWL Reasoner. This ensures automatic classification of patterns into language levels.

### 3.3 Rule-Based Inferences

Defining the criteria for pattern languages in a formal medium, such as Description Logic (DL) in OWL, facilitates a degree of utility not afforded in informal representations. In addition to logical inference, rules can be applied to the pattern descriptions and relationships to further enrich pattern languages. For example, to associate a specific list selection pattern, *TwoColumnSelectDeselect*, whenever a *ShoppingCartPattern* is chosen with a *hasSolution* instance named *userEditingSelections*, the following rule would be applied:

```
If (ShoppingCartPattern.hasSolution(userEditingSelections)
Then "include TwoColumnSelectDeselect"
```

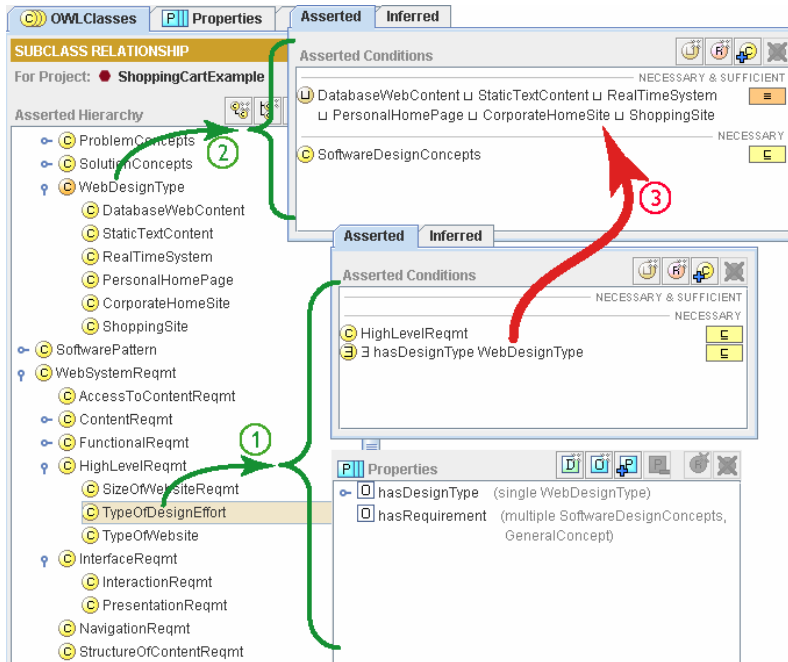


Figure 3: An Example Requirements Taxonomy.

Once part of the ontology-based pattern language definition, this rule would be executed whenever the conditions are present, either through manual (“asserted”) definition or inferred definitions.

#### 4 Specifying Project Characteristics

Formal pattern ontologies are of little use if there is no means to represent the problem domain. OWL description logic can be used to build problem domain ontologies that formally define usability requirements. We are in the process of modeling Web design usability problems using the AWARE model (Mastering the Requirements Analysis of Communication-Intensive Websites) of Website taxonomies [6] to create a representation of requirements in our proof-of-concept domain of Web shopping applications. Given a requirement ontology and the pattern ontology we have been discussing, inferences are performed through an automated reasoning process that matches requirements to patterns or other ontology components.

For the purposes of this exemplar, Figure 3 shows a sample ontology for Website requirements (the sub-tree starting at the WebSystemReqmt concept) federated with other ontologies shown in the left-hand window of Figure 3. The location of this ontology would normally be in a separate Web server and would be “imported” to a single location when performing inferences and queries. This would allow the sepa-

rate ontologies to evolve independently with experts in the respective fields acting as knowledge curators. At this point, the `TypeOfDesignEffort` concept definition has been broken out into the two windows in the brace marked ①. These definitions state that `TypeOfDesignEffort` is a type of `HighLevelReqmt` with two properties defined, `hasDesignType` and `hasRequirement` (lower pane in braces labeled ①).

The `WebDesignType` concept, a subtype of `SoftwareDesignConcepts` is defined by the window marked with brace ②. This definition shows that a `WebDesignType` concept must be defined by exactly one of its subtypes, `DatabaseWebContent`, `StaticTextContext`, and etc.<sup>2</sup> Since `TypeOfDesignEffort` is defined to have one `hasDesignType` (and only one, since it is defined to be a functional relationship, signaled by the “single” to the left of `hasDesignType` in the lower-right window of Figure 3), any instance of `TypeOfDesignEffort` will, by definition, need to specify one of the `WebDesignType` concepts as its design type (see ③). Therefore, any valid instance of `TypeOfDesignEffort` will need to be mapped to an instance of `WebDesignType` or an inconsistency in the ontology will be flagged by the Reasoner.

#### 4.1 Refining Usability Knowledge

Using the Semantic Web to deliver usability knowledge in the form of patterns benefits significantly from the accumulation and continuous refinement of pattern knowledge, where knowledge comes in the form of both pattern content and semantic relationships between patterns. We envision these repositories at the center of communities of pattern developers and such that people involved in the design and development process collaboratively construct pattern languages as new trends and technologies emerge.

Several patterns communities currently exist in various stages of sophistication. Van Welie’s pattern collection [38] has a web interface to submit comments about existing patterns. Bolchini’s pattern repository [5] has a process to submit new patterns and has about 210 active members on its mailing lists. But in both cases, key elements that relate and unify the patterns in these repositories are missing. The Semantic Web contributes the much-needed element of an intelligent and flexible way to organize and utilize the many patterns that exist within the repository. In fact, the end result of using a Semantic Web agent in the repository of patterns is one or more consistent pattern languages for usability design.

It should be clear that we do not advocate developing one centralized repository that everybody uses. Instead, we envision several repositories at several stages of sophistication. Each repository can use parts or all of the other repositories. This helps us to build pattern languages collaboratively. Although beyond the scope of this paper, the Semantic Web trust layer [28] has a great potential to provide answers to vexing problems of knowledge quality and semantic disagreement that have proven difficult for knowledge management efforts. It would be particularly fruitful to integrate these belief networks with the kind of strength of evidence and relative importance ratings used by `usability.gov` and other guideline corpora. This would allow

---

<sup>2</sup> Please note that this and other ontologies in this paper are being used as examples only and may or may not be indicative of what a community-built repository would consist of.

ratings to be based on community-wide belief systems rather than the individual opinions of experts or panels of experts.

## 5 Analysis and Contributions

One of the strengths of this work is that it utilizes a foundation of software tools that have already been developed and are in use by the Semantic Web community. OWL and DL Reasoners for OWL are recommendations by the World-Wide Web Consortium (W3C) [24]. There is a growing buy-in to these recommendations, thus ensuring that the tools will be available for years to come. Ontology development, refinement, and maintenance therefore become the primary focus of development efforts necessary to design, maintain, and refine pattern languages.

On the other hand, ontology development is notoriously difficult to understand and work with. There is no doubt that the classic “knowledge acquisition” problem is the greatest risk to the potential success of this approach. However, the network effect and the use of recently developed standard infrastructures provide some hope that this approach can succeed where others have been less than successful.

While this and other formal knowledge-based approaches are ultimately limited by the quality of the knowledge they contain, it is important to understand that this is true for all knowledge representation mediums, including books. Our approach does not seek to replace human judgment but to augment it with community-driven information that would otherwise be inaccessible or difficult to obtain. People will not seek information if they do not realize that potentially useful information exists.

We see context as one of the main organizing features of patterns. Usability issues and decisions are often, if not always, context-sensitive. Capturing this context is just the first step at making usability design a more stable and scientific endeavor. Other sources of potential disagreement and lack of widely agreed standards may have roots in personal preferences and perspectives. The research described here can enable an informed discussion of these topics by integrating current knowledge sources so they can be compared and evaluated.

## 6 Future Work

While this research is still in its formative stages, we and others believe that Web-based ontologies will play an important role in the development of next-generation Web technologies. It is therefore prudent that we begin to experiment with and develop an understanding of how evolving technologies can be harnessed to facilitate and enable improvements in knowledge-intensive fields such as usability design practices.

Formative and summative evaluation efforts will play an important role in future efforts. Immediate plans are to improve our “seed” ontologies through feedback from potential users – both pattern developers and software developers. It is anticipated that better interfaces for ontology development will be needed, as the description logic used in OWL will not be universally accessible. Through formative evalua-

tions, we hope to understand how mechanisms such as wizards, templates, and direct manipulation interfaces can be utilized to present information in terms that pattern developers use and understand while capturing semantic relationships in the background.

To be accessible to software developers, further research is needed into the utility of usability patterns and other knowledge sources in the development lifecycle. Empirical results so far have at best been mixed. Improved understanding of the issues involved and how/whether our approach can address them is clearly needed.

Beyond the critical need for evaluating various aspects of our ontology-based pattern language approach, there is a need to develop these concepts into an integrated and accessible platform for usability design support. We are currently focusing on the Eclipse platform as a framework for this effort. This has the simultaneous advantage of being an increasingly accepted platform for development and the focus of current efforts to integrate Semantic Web tools and technologies, such as Protégé or TopBraid Composer [35] into Eclipse plug-ins.

## 7 Conclusions

Instead of requiring software developers to become pattern experts on isolated collections of patterns that sparsely populate the problem and solution space, we envision a distributed repository of patterns that relate problems to solutions through typed relationships that manifests a systems design method. Both the knowledge content of these patterns and knowledge contained in semantic (typed) relationships between patterns that constitute a pattern language are created, refined and maintained by a community of experts in respective subfields.

In doing so, we have begun to resolve many of the problems that currently plague the usability patterns community, as well as the software patterns community as a whole. While a main goal of patterns is to form a vocabulary that helps developers communicate better, too many pattern collections have been created that draw little or no relationships between each other, in essence creating islands of patterns that sometimes contradict, duplicate, or are inconsistent with one another.

The objective of this research is not an attempt to completely automate user interface design. To the contrary, it is fully recognized that effective user interface design takes a degree of talent and careful work with the end users that cannot be captured through rules, patterns or any information system. This research is an exploration of how resources can be delivered to software developers through a representational medium that serves to establish relationships between context and usability resources and serves as a formal mechanism for communicating and refining usability design knowledge.

Continued research is needed to further understand the complexities of creating repositories of usability patterns and applying them proactively in the software development process. We have taken steps in this direction, and hope that future validation and use of our approach provides more information of usability knowledge and the contextual factors that impact this knowledge.

**Acknowledgements.** I gratefully acknowledge the efforts a number of students that have contributed in various ways in this and associated research, particularly Padmapriya Ashokkumar and Sarita Navuluru. This research has been funded in part by the National Science Foundation (CCF-0639164).

## References

- [1] C. Alexander, "The Origins of Pattern Theory: the Future of the Theory, and the Generation of a Living World," OOPSLA 1996 Keynote Address, <http://www.patternlanguage.com/archive/ieee/ieeetext.htm>, 1996.
- [2] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, 1977.
- [3] J. Barber, et al., "AskJef: Integration of Case-Based and Multimedia Technologies for Interface Design Support," Artificial Intelligence in Design '92, J. S. Gero, Ed(s). Kluwer Academic, pp. 457-474, 1992.
- [4] S. Bechhofer, "DL Implementation Group (DIG)," <http://dl.kr.org/dig/>, Updated: Jan., 2006.
- [5] D. Bolchini, "Hypermedia Design Pattern Repository," <http://www.designpattern.lu.unisi.ch/index.htm>, Updated: January, 2002.
- [6] D. Bolchini, Web Design Patterns: Improving Quality and Performance in Web Application Design, Communication Sciences, University of Lugano, Italy, Master Thesis, 101 pages, 2000.
- [7] J. Borchers, "CHI Meets PLoP: An Interaction Patterns Workshop," SIGCHI Bulletin, 32(1), pp. 9-12, 2000.
- [8] J. Borchers, "hcupatterns.org: Patterns," <http://www.hcupatterns.org/patterns.html>, Updated: May, 2006.
- [9] D. Brugali, K. Sycara, "Frameworks and Pattern Languages: an intriguing relationship," ACM Computing Surveys, 32(1), pp. 12-42, 2000.
- [10] I. W. Connell, N. V. Hammond, "Comparing usability evaluation principles with heuristics," Proc. INTERACT, pp. 621-636, 1999.
- [11] W. Cunningham, "Portland Pattern Repository," <http://c2.com/ppr/>, Updated: Sept., 2006.
- [12] F. de Souza, N. Bevan, "The Use of Guidelines in Menu Interface Design: Evaluation of a Draft Standard," Human-Computer Interaction - INTERACT '90, Elsevier, North-Holland, pp. 435-440, 1990.
- [13] J. Deng, E. Kemp, E. G. Todd, "Managing UI pattern collections," Proc. 6th ACM SIGCHI New Zealand Chapter's Int'l Conf. on Computer-Human Interaction (CHINZ '05), pp. 31-38, 2005.
- [14] T. Erickson, "The Interaction Design Patterns Page," <http://www.visi.com/~snowfall/InteractionPatterns.html>, Updated: June, 2005.
- [15] T. Erickson, "Lingua Francas for Design: Sacred Places and Pattern Languages," Proc. Designing Interactive Systems (DIS 2000), New York, pp. 357-368, 2000.
- [16] S. Fincher, "CHI 2003 Workshop Report - Perspectives on HCI patterns: concepts and tools (introducing PLML)," Interfaces, 56, pp. 26-28, 2003.
- [17] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.
- [18] J. D. Gould, C. H. Lewis, "Designing for Usability - Key Principles and What Designers Think," Communications of the ACM, 28, pp. 300-311, 1985.
- [19] I. Graham, A Pattern Language for Web Usability. Addison-Wesley, 2003.

- [20] S. Henninger, K. Haynes, M. W. Reith, "A Framework for Developing Experience-Based Usability Guidelines," Proc. Designing Interactive Systems (DIS '95), Ann Arbor MI, pp. 43-53, 1995.
- [21] S. J. Koyanl, R. W. Bailey, J. R. Nall, "Research-Based Web Design & Usability Guidelines," Communications Technology Branch, National Cancer Institute & US Dept of health and Human Services, <http://www.usability.gov/pdfs/guidelines.html>, 2003.
- [22] E. Lai-Chong Law, E. T. Hvannberg, "Analysis of Strategies for Improving and Estimating the Effectiveness of Heuristic Evaluation," Proc. 3rd Nordic Conf. on HCI, pp. 241-250, 2004.
- [23] D. L. McGuinness, F. van Harmelen, "OWL Web Ontology Language Overview," W3 Consortium, <http://www.w3.org/TR/owl-features/>, Updated: February 10, 2004.
- [24] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, G. Schreiber, D. Wood, D. Connolly, "W3C Semantic Web," World-Wide Web Consortium, <http://www.w3.org/2001/sw/>, Updated: March 15, 2006.
- [25] S. Montero, P. Díaz, I. Aedo, "A Semantic Representation for Domain-Specific Patterns," in Int'l Symp. on Metainformatics, U. K. Wiil, Ed., Springer-Verlag, LNCS 3511, 2005, pp. 129-140.
- [26] J. Noble, "Towards a Pattern Language for Object-Oriented Design," Proc. of Technology of Object-Oriented Languages and Systems (TOOLS Pacific), 28, IEEE Computer Society Press, pp. 2-13, 1998.
- [27] PLoP, "PatternLanguagesOfPrograms," Hillside.net, <http://hillside.net/plop/>, 2005.
- [28] M. Richardson, R. Agrawal, P. Domingos, "Trust Management for the Semantic Web," Proc. 2nd Int'l Semantic Web Conference (ISWC), Springer, pp. 351-368, 2003.
- [29] S. L. Smith, J. N. Mosier, "Guidelines for Designing User Interface Software," ESD-TR-86-278, Technical Report, The MITRE Corporation, 1986.
- [30] Stanford Univ., "Protégé Project," Stanford Medical Informatics, <http://protege.stanford.edu/>, Updated: Aug. 10, 2006.
- [31] R. Studer, V. R. Benjamins, D. Fensel, "Knowledge Engineering: Principles and Methods," Data and Knowledge Engineering, 25, pp. 161-197, 1998.
- [32] L. Tetzlaff, D. R. Schwartz, "The Use of Guidelines in Interface Design," Proc. Human Factors in Computing Systems (CHI '91), ACM, New York, pp. 329-333, 1991.
- [33] H. Thovtrup, J. Nielsen, "Assessing the usability of a user interface standard," Proc. Human Factors in Computing Systems (CHI '91), New Orleans, LA, pp. 335-341, 1991.
- [34] J. Tidwell, "UI Patterns and Techniques," <http://time-tripper.com/uipatterns/>, Updated: May, 2005.
- [35] TobBraid, "TopBraid Composer," TopQuadrant, Inc., <http://www.topbraidcomposer.com/>, Updated: Aug. 8, 2006.
- [36] E. Todd, E. Kemp, C. Phillips, "What makes a good User Interface pattern language?," 5th Australasian User Interface Conf. (AUIC2004), Australian Computer Society, Inc., pp. 91-100, 2004.
- [37] D. K. van Duyne, J. A. Landay, J. I. Hong, The Design Of Sites. Addison-Wesley, 2002.
- [38] M. van Welie, "Patterns in Interaction Design," <http://www.welie.com/>, Updated: June 27, 2006.
- [39] M. van Welie, G. C. van der Veer, "Pattern Languages in Interaction Design: Structure and Organization," Proc. Interact '03, M. Rauterberg, Wesson, Ed(s). IOS Press, Amsterdam, The Netherlands, Zürich, Switzerland, pp. 527-534, 2003.
- [40] Yahoo!, "Yahoo! Design Pattern Library," <http://developer.yahoo.com/ypatterns/>, 2006.