2005

Disseminating Usability Design Knowledge through Ontology-Based Pattern Languages

Scott Henninger
University of Nebraska-Lincoln, scotth@cse.unl.edu

Padmapriya Ashokkumar
University of Nebraska-Lincoln, ashokkum@cse.unl.edu

Disseminating Usability Design Knowledge through Ontology-Based Pattern Languages

Scott Henninger
Univ. of Nebraska-Lincoln
Computer Science and Eng.
Lincoln, NE 68588-0112
+1-402-472-8394
scotth@cse.unl.edu

Padmapriya Ashokkumar
Univ. of Nebraska-Lincoln
Computer Science and Eng
Lincoln, NE 68588-0112
ashokkum@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. Through this method, loosely coupled pattern collections can be turned 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.

Categories and Subject Descriptors
D.3.3 [**Patterns**]: Language Constructs and Features – *usability patterns.*

General Terms
Documentation, Design, Human Factors.

Keywords
Pattern languages, usability guidelines, Semantic Web, user interface design.

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 or usability issues. Education, use of HCI specialists in the development process, and iterative development processes aimed at evaluating and improving the user interfaces [21] 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. Studies on the application of usability guidelines have had mixed results, with some demonstrating that both novice and expert HCI specialists benefit from guidelines [14, 26], whereas others have revealed challenges with finding and applying guidelines for specific problem settings [36, 37]. However, all have found significant problems with the manner in which the knowledge is disseminated and applied.

Current approaches to representing usability knowledge are document-based, at best supported with hypertext tools and/or in Web pages. 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 our research are to 1) build tools that puts context-appropriate usability guidelines at the fingertips of software designers and usability specialists so they can be used early and throughout in the design and development process, 2) construct a formal computational framework for creating interconnected corpora of usability knowledge, and 3) provide a Web-based infrastructure to facilitate community-based evolution of usability knowledge and the contexts in which specific usability resources – techniques, principles, guidelines, and etc. are most effective.

In this paper, we present a framework in which usability patterns [9, 19] are used for representing usability knowledge and Semantic Web ontologies [27] are used to formally define pattern attributes and relationships between the patterns. 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 and 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, we first describe usability patterns and the types of tools and support currently available for applying usability and other software development guidelines and patterns. Some general background is given on using Semantic Web technologies to implement pattern languages followed by a specific example of ontologies and associated rules and inferences that allow intelligent support for applying usability patterns.

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 [6, 25, 33]. Usability guidelines provide principles and concepts that lead to good interface design from both general and widget-specific perspectives. While hundreds of usability guidelines have been designed and published, empirical studies have demonstrated a number of difficulties in understanding and applying guidelines [36] and the difficulty of determining when a guideline has been violated [37]. These and other problems can be seen to stem from the abstract and decontextualized nature of current guideline techniques [15, 24]. This creates 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" [4].

A usability patterns community [10, 22, 41, 44], inspired by work on software design patterns [20], has begun to explore how patterns can be used to provide enhanced

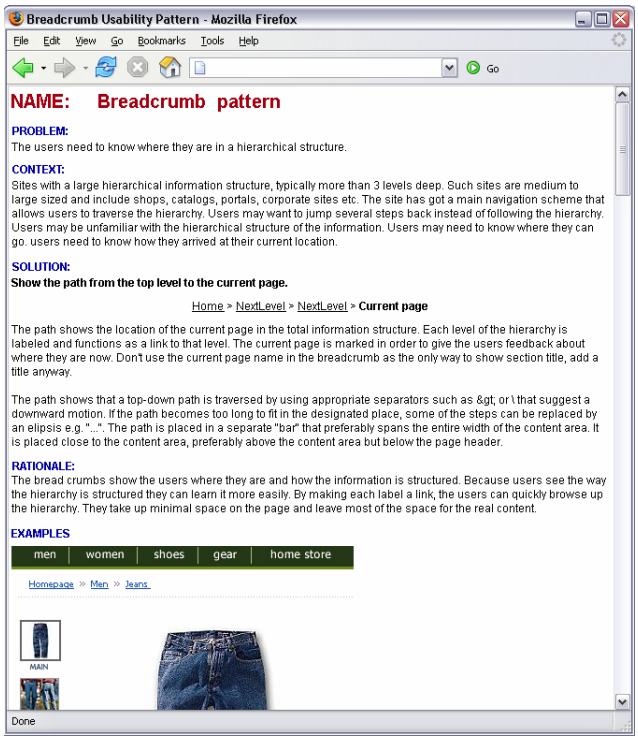

Figure 1: The Breadcrumb Pattern.

representation for usability knowledge that explicitly defines the context and interrelationships between patterns [18]. The essential idea of a design pattern is to capture successful solutions to recurring problems along with the context and forces that operate on the problem to yield a general, repeatable, solution [2, 3].

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, than the more general information or advice perspective of guidelines. As shown by the Breadcrumb pattern [43] in Figure 1, patterns also add fields to explicitly describe the context of the problem and the forces that shape the problem and its variants (see the XML Schema-based Pattern Language Markup Language (PLML) [19] as an example). 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.

2.1 Pattern Collections and Pattern Languages

The majority of existing Patterns are organized in collections of loosely coupled sets of Pattern descriptions classified by defined criteria [20, 25] or a taxonomy [43]. Each Pattern of the collection is typically presented in the same uniform format for better readability and understandability. These collections tend to be self-contained "islands" of knowledge that rarely contain

pointers outside of their boundaries. Collections of patterns can be organized in a network of higher-level patterns that are resolved or refined by more detailed patterns, resulting in a *Pattern Language* [3, 45].

While the original pattern work by Christopher Alexander for Architectural design defined pattern languages as generating holistic design solutions [3], this perspective has largely been lost when applied to software patterns [1]. When discussing collections of patterns, current literature either provides a murky definition of pattern collections or uses the concept interchangeably with pattern languages. We wish to make a clear distinction between pattern *collections* and pattern *languages* along two dimensions. 1) While pattern collections are relatively isolated, pattern languages are highly interconnected [40]. This leads to more robust knowledge structures with a higher probability of filling in the gaps that allow 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 complete design solutions. 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 sequence of decisions [12], levels of scale (architecture), and other forms that aim toward the orderly resolution of design processes through the systematic design and reuse of patterns.

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 [10, 17], Wiki pages [30, 31], and books. Pattern representations are document-based, at best supported with hypertext tools and/or Web browsing [7, 42, 43]. These passive representations rely on individual developers to know of the existence of the resources and understand when they should be applied. Given the potentially copious numbers of patterns 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 application that can effectively support design processes.

Suppose a project team is developing 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. But this leaves the sticky problem of how to collect items that have been chosen in different places in the site, both from a usability perspective and an information retention perspective (i.e. keeping track of chosen items across separate Web pages).

Some members of the team are aware that proven usability knowledge for these types of interfaces is available and refer to the Interaction Design Patterns website (often referred to as the Amsterdam Patterns Collection) [43] containing over 60 usability patterns, including guidelines relevant to the project such as Ecommerce and Web shopping patterns. Many other pattern collections exist, both in Web sites [23, 38, 39] and books [11, 41], and could also be used by the team instead of or in addition to this pattern collection.

Given the discovery of this pattern collection, the team must read, digest and sort out the collection of patterns to find which ones might be applicable on 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 [43] 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 conjunction 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) need to 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 the needs for certain types of search interactions. The patterns are represented in informal natural language, at best using hyperlinks, or a "Related patterns" field that link to other patterns in the collection, again leaving the interpretation of this single type of relationship to the pattern user. Therefore, 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 [39], 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. THE SEMANTIC WEB 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 the context field to formally or systematically indicate when a pattern should be used, has yet to be explored in any detail. Alexander stated that "Each pattern is a three part rule, which expresses a relation between a certain context, a problem, and a solution." [2];p. 247. In the following sections, we describe how ontological descriptions using standardized Semantic Web technologies can be utilized to provide typed relationships between patterns.

The Semantic Web is gaining widespread acceptance as a Web-based knowledge delivery technology [28]. It supports formal descriptions of information in a computational format for machine processing that can easily be converted into human-readable forms [5]. 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. 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 a pattern language
- automatically and dynamically classify patterns into pattern languages that can generate complete design solutions
- check the consistency of patterns and pattern language attributes

After defining some of the key concepts used in this approach, OPAL (Ontology based PAttern Languages) is presented as an example of a tool that enables the delivery of pattern languages to everyday software developers 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" [35]. An ontology is created as a set of definitions from a formal vocabulary defining a "schema" and instances (often referred to as individuals) of the schema concepts. In addition, the Semantic Web utilizes Uniform Resource

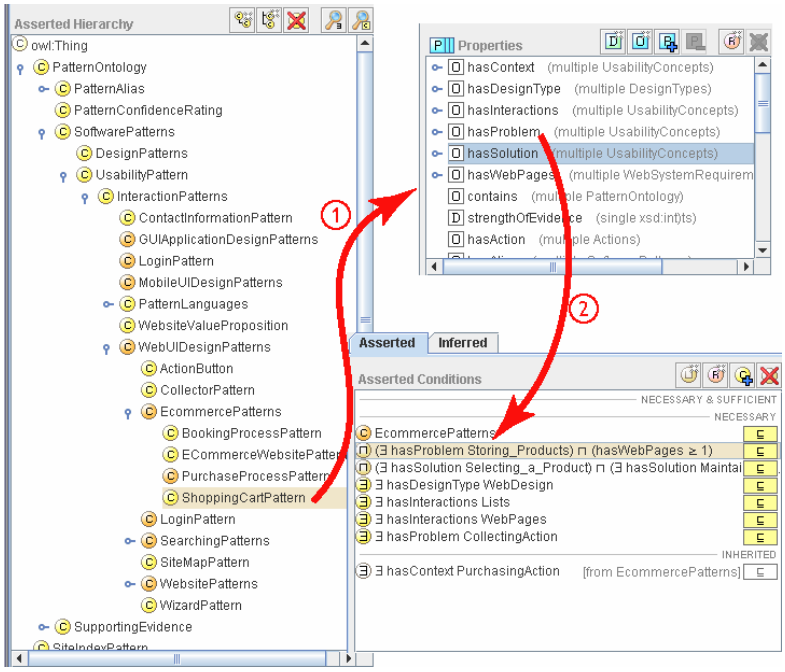

Figure 2: An OWL description of a usability design pattern.

Indicators (URIs), a generalization of URLs, to create unique namespaces in spite of being distributed across the WWW. When creating an ontology, one can choose concepts from different ontologies in different file on different servers. Each choice represents a form of "ontological commitment" [29] that in essence states that the new ontology is in agreement with the chosen ontology on that term. Therefore, if all OWL files defining usability pattern concepts refer to the same URI for the ShoppingCartPattern[1] (or, for example, a concept with an *equivalentClass* relationship) then all ontologies are guaranteed to computationally refer to the same concept.

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 a standard Reasoner such as Racer, FaCT, or Jena [13] to classify and to make inferences (examples will be given later in the paper). Figure 2 contains a screen image of parts of an ontology developed in an ontology editor, named Protégé 3.1 [34], using the Semantic Web language OWL (Web Ontology Language). OWL implements forms of first-

[1] As a concrete example of a URI, the ontology used for examples in this proposal is stored on-line at http://cse.unl.edu/~scotth/SWont/ShoppingCartExample.owl. A direct URI to the ShoppingCart concept in that ontology is: http://cse.unl.edu/~scotth/SWont/ShoppingCartExample.owl#ShoppingCartPattern.

order predicate logic [28] to define concepts through restrictions on concepts and properties.

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 [19] to include additional properties for reasoning, strength of evidence for the pattern and other pattern attributes.

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 (∃ hasProblem Storing_Products) ∩ (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 (∃). This is joined with a logical AND (intersection in OWL) with the statement that there must be more than one hasWebPages definition.

The hasContext property of the ShoppingCartPattern can also be defined formally in OWL so that the context this pattern is used in can be computed and matched against requirements concepts in OPAL (see below). One of the design contexts for Shopping Cart is that it is used in web interface design. This is represented using the restriction (∃ hasDesignType WebDesign) in Figure 2. Since hasDesignType is a subproperty of hasContext, this implies that Shopping Cart Patterns is used in the context of WebDesign. Apart from Design Type, has Context has other subproperties like SiteType, UserExperiences, size, etc. which can be used to describe contexts of this and other patterns.

Describing pattern attributes in such a formal manner helps us to infer relationships between patterns instead of manually defining them as is done in text based pattern languages. For example, any pattern that has the same

(∃ hasSolution Selecting_products) and

(∃ hasSolution maintaining_list_of_products) and

(∃ hasSolution storing_lists_of_products_on_client_side) or
(∃ hasSolution storing_lists_of_products_on_server_side)

Table 1: OWL statements for the solution property of a ShoppingCartPattern.

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 Necessary and Sufficient conditions, a Reasoner can infer classification. Although not shown here, if the restriction "∃ hasDesignType WebDesign" were stated as the only Necessary and Sufficient condition 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 for a pattern *language*. By our definition, a software pattern language consists of a collection of patterns with a structuring principle that can help perform a complete design for a specific domain of software systems. There are many possible types of structuring principles aimed at the orderly resolution of patterns. 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 [45] that defines successive levels of problem decomposition. Four levels of decomposition are defined –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, Widget 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." [45]. This can be formally stated as:

(hasSeries ≥ 1) ∩ (∃ hasSeries Actions) ∩ (∀ hasSeries Actions) ∩ (∃ hasObject UsabilityObject) ∩ (hasObject ≥ 1).

Informally, this states that a task level pattern is any pattern that has at least one series of Actions on at least one usability object. The above restriction is described in OPAL as a Necessary and Sufficient condition, meaning that any pattern that satisfies these conditions will be classified as a task level pattern by a Reasoner. This ensures automatic classification of patterns into language levels.

3.3 Benefits of Computational Pattern Languages

Defining the criteria for pattern languages in a formal medium, such as description logic in the Semantic Web, facilitates a degree of utility not afforded in informal representations. Firstly, in our ontological definition of patterns, we assume that patterns are defined independently of pattern languages. This allows pattern designers to define patterns based on the pattern's characteristics and allows patterns to be used in multiple pattern languages. The necessary and sufficient conditions for each level can be used to classify patterns into the appropriate language levels. For example the solution for a ShoppingCartPattern is described formally using the OWL statements in Table 1. Since Selecting_products is a type of Selection and maintaining_list_of_products is a type of Maintanence both of which are types of Interactions, it can be inferred that ShoppingCartPattern has a series of Interactions. Also, since Task Level patterns are those patterns that have a series of interactions it is inferred further that ShoppingCartPattern is a Task Level pattern.

The significance of this classification is that patterns can be part of pattern languages by satisfying the criteria defined in the pattern language levels. This improves the flexibility and extensibility of the pattern languages, particularly when defined in an open world-wide medium such as the infrastructure defined by Semantic Web technologies. By defining the necessary and sufficient conditions for class membership, patterns defined as ontology concepts can become part of any language where the specified conditions are met. Different pattern experts need to look at pattern collections with different perspectives. It is not feasible to expect everybody to follow one pattern language or classification scheme. Allowing multiple classification schemes accommodates the wide variety of pattern experts. The implications of these languages and inferences for

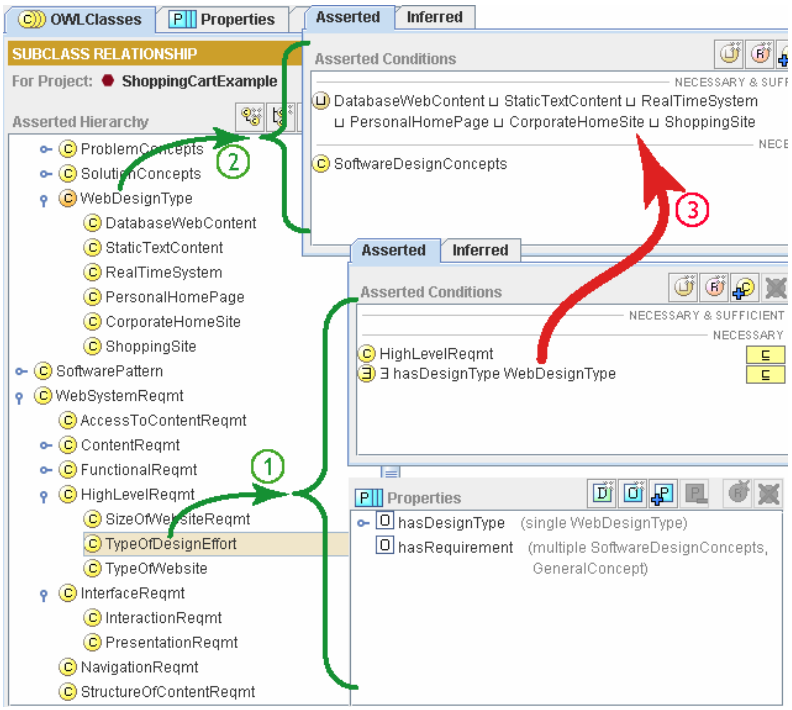

Figure 3: OPAL Requirements Taxonomy.

pattern-based development include defining choices that can be made while traversing a '*levels of decomposition*' pattern language. For example, choosing an interface pattern that requires a browsing method could define the conditions for facilitating information browsing. Then any pattern concept meeting the criteria will match and become a choice for browsing that appears in a list of patterns to select from.

Further, given the language criteria that patterns at all levels of abstraction must be connected in the language, it becomes a computational exercise to verify that this is true and that other inconsistencies do not exist. This provides a much higher level of assurance that using the language will lead to satisfactory results than informal text-based pattern languages where either the relationships may not be complete or worse may be contradictory.

This is just one example of how OWL-DL can be used to formally describe usability pattern languages. Given this baseline, number of classification and consistency checking inferences are available to further refine pattern languages. In addition to these logic-based inferences, 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"

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. USING OPAL TO SPECIFY 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. OPAL uses the AWARE model (Mastering the Requirements Analysis of Communication-Intensive Websites) of Website taxonomies [8] to create a representation of requirements in our proof-of-concept domain of Web shopping applications. Given a requirements 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 OPAL 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 separate 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 and WebDesignType concept must be defined by exactly one of its subtypes, DatabaseWebContent, StaticTextContext, and etc.[2] 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 it's 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.

[2] 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.

4.1 Collecting Project-Relevant Patterns

The objective of using a pattern language is to match an appropriate, consistent, and comprehensive set of usability patterns to the specific needs of a usability development effort. Figure 4 shows how OPAL collects usability patterns as they are matched to project characteristics. Whenever a pattern is added to the team's working environment, OPAL is used to choose whether to include these patterns depending on project requirements. Including a pattern into the project environment means that a design decision has been made to follow the solution outlined in the included pattern.

After some of the high level options have been chosen, the team has a set of patterns for the overall design of the site. Note that options can be chosen in any order at any time during the development effort. In addition, previous options can be modified to meet emergent and changing project needs. At the point depicted in Figure 4, project personnel need to customize some of these patterns to suit their specific lower level requirements and also identify what lower level patterns are required to complete the high level design. In the traditional pattern collection (in books and web) method they would have to go through another iteration of reading the pattern collection to find relevant patterns. In our system the high level patterns have options and choices that help customize the patterns along with the relationships to lower level patterns in the pattern language. For example, the Shopping Cart Pattern has certain options like, "persistent images that links shopping cart from every page", "stored on client side using cookies", "stored on server side". Choosing the option "persistent images that links shopping cart from every page leads" to a "Fast downloadable image" pattern. Also the tree hierarchy represents patterns from high level to low level thus helping the designer traverse through the pattern language. A human readable version of each pattern is created by using the XML form of the OWL definitions and rendering a Web page. For example, Figure 1 shows the Breadcrumbs pattern as it has been rendered from the OPAL ontology.

In addition, the design team can choose between alternative patterns at a certain level of the pattern language. For example, List Builder pattern or Wizard pattern are task level patterns that can be used to implement a Shopping Cart pattern. This option represents the design decisions that the team needs to make. Choosing one or the other option prunes the tree and removes patterns related to the selection not chosen. This ensures that irrelevant patterns are removed from the designers working environment. This saves the designer from having to read every pattern to know what he wants to use and when.

4.2 Refining Usability Knowledge

The idea of using the Semantic Web to deliver usability knowledge in the form of patterns is based on the

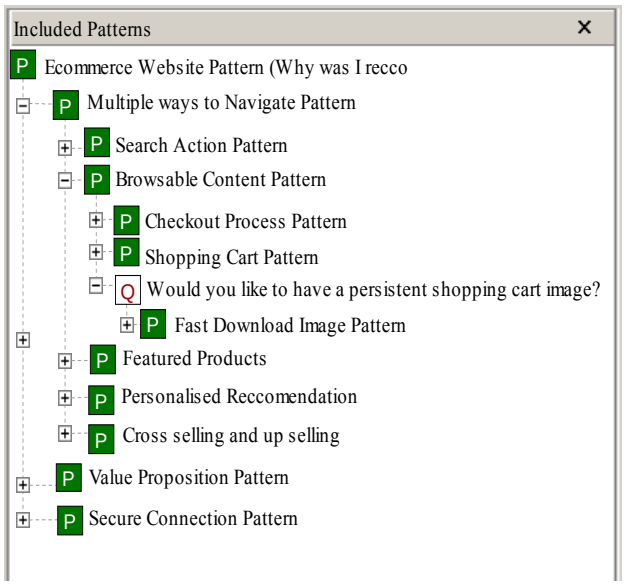

Figure 4: A Project Pattern Set In OPAL.

repository of patterns available. This repository is designed to be the center of a community of pattern developers and users that continuously evolves to create new knowledge that benefits the community. Thus, people involved in the design and development process collaboratively construct the pattern language as new trends and technologies emerge.

Several such patterns communities currently exist in various stages of sophistication, but they are simply places to store patterns. Van Welie's pattern collection [43] has a web interface to submit comments about existing patterns. Bolchini's pattern repository [7] has a process to submit new patterns and has about 210 active members on its mailing lists but the key element of intelligence that can relate and unify the patterns in these repositories is 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.

We should be clear that we are not developing one centralized repository which everybody uses. Instead there are several repositories at several stages of sophistication and 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 propagates peer-based trust ratings (i.e. "which knowledge stores and/or concepts does one believe in") to create individualized networks of belief [32]. This 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 ratings to be based on community-wide belief systems rather than the individual opinions of experts or panels of experts.

5. ANALYSIS AND EVALUATION

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 Reasoners for OWL are recommendations by the World-Wide Web Consortium (W3C) [28], the same standards organization that developed and maintains HTML, XML and other standards. There is a growing buy-in to these recommendations, thus ensuring that the tools will be available for years to come. This leaves ontology development, refinement, and maintenance as the primary development effort necessary for OPAL to develop 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 it contains, 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.

This brings up another knowledge quandary in need of further investigation. By letting the reasoner infer relationships between patterns we have a much higher chance of finding relationships, representations of usability knowledge that the pattern designer did not realize existed. But as the pattern language grows this becomes a non-trivial task as it involves more effort on the part of the pattern designer to find relationships of a particular pattern with other patterns in the language.

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. This work 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 formative stages, we feel early dissemination will both bring some important issues to the attention of the HCI community while helping to jump start the community of potential user and evaluators that will be necessary to mature and refine this technology. We and others believe that the Semantic Web 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 improved usability design practices.

Much of the work presented here has yet to be held to the scrutiny of validation efforts. Many questions remain unanswered for both Semantic Web technologies and the use of ontologies to effectively disseminate patterns through formally defined languages. Formative and summative evaluation efforts will play 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 usability evaluations, 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. Improves understanding of the issues involved and how/whether our approach can address them is clearly needed.

Beyond the critical need for evaluating various aspect of our ontology-based pattern language approach, there is a need to port the current OPAL prototype from its currently fractured state into a more integrated and accessible platform. We are currently focusing on the Eclipse [16] platform as a framework for OPAL. 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 the Protégé ontology tool used in this work, 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 and is 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. Nonetheless, there is recognized knowledge and conventions that can help some designers reach higher levels of competency and help accomplished designers extend their knowledge to areas they have not yet experienced. 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.

8. ACKNOWLEDGMENTS

We gratefully acknowledge the efforts of a number of students that have contributed in various ways in this and associated research, including Ryan Kinworthy and Sarita Navuluru. This research was funded in part by the National Science Foundation (CCR-9988540).

9. 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, The Timeless Way of Building. Oxford Univ. Press, New York, 1979.

[3] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, 1977.

[4] J. e. a. Barber, "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.

[5] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web," Scientific American, May 2001, 2001, on-line at: http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2.

[6] P. A. Billingsley, "Starting from Scratch: Building a Usability Program at Union Pacific Railroad," interactions, 2(4), pp. 27-30, 1995.

[7] D. Bolchini, "Hypermedia Design pattern Repository," http://www.designpattern.lu.unisi.ch/index.htm, Last Updated January, 2002.

[8] D. Bolchini, Web Design Patterns: Improving Quality and Performance in Web Application Design, Communication Sciences, University of Lugano, Italy, Master Thesis, 101 pages, 2000.

[9] J. Borchers, "CHI Meets PLoP: An Interaction Patterns Workshop," SIGCHI Bulletin, 32(1), pp. 9-12, 2000.

[10] J. Borchers, "hcipatterns.org: Patterns," http://www.hcipatterns.org/patterns.html, Last Updated March 2004, 2004.

[11] J. Borchers, A Pattern Approach to Interaction Design. John Wiley & Sons, 2001.

[12] D. Brugali, K. Sycara, "Frameworks and Pattern Languages: an intriguing relationship," ACM Computing Surveys, 32(1), pp. 12-42, 2000.

[13] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, "Jena: implementing the semantic web recommendations," Proc. 13th Int'l World Wide Web Conf., pp. 74-83, 2004.

[14] I. W. Connell, N. V. Hammond, "Comparing usability evaluation principles with heuristics," Proc. INTERACT, pp. 621-636, 1999.

[15] 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.

[16] Eclipse, "eclipse.org," http://www.eclipse.org/, Last Updated Sept. 8, 2005.

[17] T. Erickson, "The Interaction Design Patterns Page," http://www.visi.com/~snowfall/InteractionPatterns.html, Last Updated March 2005, 2005.

[18] T. Erickson, "Lingua Francas for Design: Sacred Places and Pattern Languages," Proc. Designing Interactive Systems (DIS 2000), New York, pp. 357-368, 2000.

[19] S. Fincher, "Perspectives on HCI patterns: concepts and tools (introducing PLML)," Interfaces, 56, pp. 26-28, 2003, on-line at: http://www.bcs-hci.org.uk/interfaces.html.

[20] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.

[21] J. D. Gould, C. H. Lewis, "Designing for Usability - Key Principles and What Designers Think," Communications of the ACM, 28, pp. 300-311, 1985.

[22] I. Grahm, A Pattern Language for Web Usability. Addison-Wesley, 2003.

[23] R. Griffiths, "The Brighton Usability Pattern Collection," http://www.cmis.brighton.ac.uk/research/patterns/home.html, 2002.

[24] 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.

[25] 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.

[26] 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.

[27] D. L. McGuinness, F. van Harmelen, "OWL Web Ontology Language Overview," W3 Consortium, http://www.w3.org/TR/owl-features/, Last Updated February 10, 2004.

[28] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, G. Schreiber, D. Connolly, "W3C Semantic Web," World-Wide Web Consortium, http://www.w3.org/2001/sw/, Last Updated Aug. 19, 2005.

[29] N. F. Noy, D. L. McGuiness, "A Guide to Creating Your First Ontology," http://protege.stanford.edu/publications/ontology_development/ontology101.pdf, Last Updated June, 2002.

[30] PLoP, "PatternLanguagesOfPrograms," Hillside.net, http://hillside.net/plop/, 2005.

[31] Portland, "Portland Pattern Repository," http://c2.com/ppr/, Last Updated June 15, 2005.

[32] 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.

[33] S. L. Smith, J. N. Mosier, "Guidelines for Designing User Interface Software," ESD-TR-86-278, Technical Report, The MITRE Corporation, 1986.

[34] Stanford Univ., "Protégé Project," National Library of Medicine, http://protege.stanford.edu/, Last Updated Feb. 2005, 2005.

[35] R. Studer, V. R. Benjamins, D. Fensel, "Knowledge Engineering: Principles and Methods," Data and Knowledge Engineering, 25, pp. 161-197, 1998.

[36] 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.

[37] 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.

[38] J. Tidwell, "COMMON GROUND: A Pattern Language for Human-Computer Interface Design,"

http://www.mit.edu/~jtidwell/common_ground.html, Accessed: June. 2005, Last Updated June 1999, 1999.

[39] J. Tidwell, "UI Patterns and Techniques," http://time-tripper.com/uipatterns/, Last Updated May, 2005.

[40] E. Todd, E. Kemp, C. Phillips, "What makes a good User Interface pattern language?," 5th Australasian User Interface Conference (AUIC2004), Australian Computer Society, Inc., pp. 91-100, 2004.

[41] D. K. van Duyne, J. A. Landay, J. I. Hong, The Design Of Sites. Addison-Wesley, 2002.

[42] D. K. van Duyne, J. A. Landay, J. I. Hong, "Design of Sites Pattern Browser," Addison-Wesley, http://www.designofsites.com/pb/, 2003.

[43] M. van Welie, "Patterns in Interaction Design," http://www.welie.com/, Last Updated 25-05-2005, 2005.

[44] M. van Welie, G. van der Veer, A. Eliens, "Patterns as Tools for User Interface Design," Workshop on Tools for Working With Guidelines, Biarritz, France, 2000.

[45] M. van Welie, G. C. van der Veer, "Pattern Languages in Interaction Design: Structure and Organization," Proceedings of Interact '03, Zürich, Switserland, M. Rauterberg, Wesson, Ed(s). IOS Press, Amsterdam, The Netherlands, pp. 527-534, 2003.

[46] S. Weinschenk, P. Jamar, S. C. Yeo, GUI Design Essentials. Wiley, New York, 1997.