Introducing new Pattern Language Concepts and an Extended Pattern Structure for Ubiquitous Computing Application Design Support

René Reiners
Fraunhofer Institute for Applied Information Technology FIT
Sankt Augustin, Germany
rene.reiners@fit.fraunhofer.de

Irina Astrova
Institute of Cybernetics
Talinn University of Technology
Talinn, Estonia
irina@cs.ioc.ee

Alfred Zimmermann
BIRC - Business Informatics RC
Reutlingen University
Reutlingen, Germany
alfred.zimmermann@reutlingen-university.de

***Abstract*—The method of collecting and communicating design knowledge in the shape of design patterns is a proven method, which is applied in different domains originating from architecture over software engineering, organizational aspects up to application design. This work introduces an extension of the pattern language concept and the applied pattern format introduced by Christopher Alexander, Jan Borchers and others. Our intention is to formulate a generic pattern language for ubiquitous computing application design. The language should enable contributors to integrate ideas and approaches from related domains into one pattern language construct. We present new and extended features for the "traditional" pattern language concept and its pattern structures on a conceptual basis. We also address needed features when opening the pattern language concept for a community that is asked to provide feedback to given patterns and contribute new findings.**

***Keywords*-Design Patterns, Pattern Language, Extension, Community, Collaboration**

I. INTRODUCTION

Mark Weiser's vision of *ubiquitous computing (UbiComp)* has motivated and influenced the development and implementation of smart embedded devices [1]. In the recent decades, their physical size has been significantly reduced accompanied by novel interaction concepts for smart environments. Currently, different industrial and research approaches enter different distribution channels creating a huge diversity of concepts concerning application design, service provisioning, and interaction techniques. The knowledge and experience from these approaches, however, is currently inherent to the individual systems and concepts or focusing on specific problems, respectively. Thus, general design guidelines derived from proven realizations are hard to reveal, capture, and transfer to other application designs.

As a consequence, proven concepts are currently hard to compare or combine. If the latter was possible, recommendations and ratings for often used combinations of good design practices could be realized. Well working concepts and designs are usually elaborately documented. Bad practices or failing concepts can also be valuable to avoid design mistakes a priori. There is a manifold of possibilities about what to offer and how to offer smart services in UbiComp environments [2]. Accessing these services and the way of interacting with them can also be very different. Examples for smart services offered in a UbiComp environment can be found in [3], [4], [5], [6], [7], [8]. Currently, smart services of different kinds have found their way to mobile devices. Mostly, they are independently encapsulated into small applications for current smart phones.

Looking at the potential for new ways of service provisioning and interaction, there is a danger for the users in this new world: *Disorientation*. In a new environment, how should they know what kind of smart services is available and how to access them? Maybe there is already experience with another system. However, this knowledge does not help in the current situation. Therefore, solutions need to be found that are generally applicable to many different situations and kinds of services and applications, respectively. This work seeks for methods to structure application design knowledge from current as well as future approaches and applications in the domain of ubiquitous computing. The structure is intended to serve as a repository containing guidelines supporting application designers in the domain of ubiquitous computing environments. Additionally, design flaws are rarely published, leaving the danger that they can potentially be repeated. This kind of design knowledge also needs to be preserved and made available together with working design guidelines. So, the central question of our work is formulated as follows:

"How can we abstract, combine, and keep alive UbiComp application design knowledge?"

II. PATTERN LANGUAGES AS APPROACH TO A SOLUTION

In [2], we described first steps towards an approach to gather, generalize, structure and manage UbiComp application design patterns. The first step was to find a common denomination of concepts that are used in different approaches and domains but still have common roots. Smart Services, Smart Devices and Smart Environments were introduced as well as an optional *take-away attribute* for smart services

 ISBN: 978-1-61208-158-8

allowing for remotely controlling smart services that are normally bound to objects and locations in the real world.

After the generalization of terms and concepts in application design, we need to find a *structure* for design practices. *Pattern Languages* provide a very flexible and extensible way to describe proven design solutions (cf. section III). The presented conceptual basis allows for constructing a first subset of patterns and the relations between them.

III. Related Work

Recurring design problems are a well-known phenomenon affecting many different domains. Not only in the technical sector but also in design areas like architecture, construction or user interface design, people see themselves faced with problems for which they do not directly know a solution.

However, it may happen that someone else has also treated the same or a similar problem and that a solution or guideline was found. In a good case scenario, this best practice was written down or kept in a way that it could be shared and distributed. Thus, a pattern represents knowledge for a specific domain, discusses the context in which it could be applied and explains ways to solve a certain problem.

Current pattern collections originating in the field of architecture [9], and being extending predominantly by Borchers in interactive exhibit design [10], up to the design of websites [11] or HCHI application design [12] mostly go even further. On the one hand, they can be distinguished by the vocabulary used for explaining a solution (e.g., very technical vs. descriptive and better suited to non-domain experts). On the other hand, many collections organize and structure particular units of information in a way that they can be organized hierarchically thus having different degrees of information detail.

Hierarchies, for example, can make use of the spatial dimension [9] or the pattern's level of technical description detail [12]. Others remain on a very technical level but then structure the patterns by purpose like in Gamma et al. in their collection of software design patterns [13].

Besides this informal way of describing design solutions, semiformal and formal approaches were developed. The three different concepts are briefly discussed in the following.

A. Informal Representation of Design Patterns

HCI and application design patterns are traditionally represented in natural language and stored in loosely coupled documents. Although there are different formats for such documents (the most common is a canonical form), the documents usually contain a set of fields describing the context in which a problem occurs, the intent of the design pattern, the relevant forces that justify why the design pattern and its implementation should be used and how to generate the solution for the problem. The term description seems more suitable for this type of representation.

Although informal representation helps users to understand the rationale behind a design pattern, there is a limitation to precision due to the use of natural language and the semantic ambiguity [14]. This disadvantage does not allow for any level of automation to resolve widely recognized problems such as finding and selecting the appropriate design patterns or applying them [15].

B. Semiformal Representation of Design Patterns (UML)

Several attempts have been made to represent design patterns in a semiformal way. Most of these attempts [15], [16], [17], [18] are based on UML (Unified Modeling Language [19]) descriptions. UML helps in specifying the structural and behavioral aspects of design patterns using class/object and sequence/collaboration diagrams, respectively. However, UML does little to help users understand the intent, applicability and consequences of design patterns [15].

C. Formal Representation of Design Patterns

As an attempt to resolve the problems above, different approaches were proposed to formalizing representation of design patterns. These approaches are based on either pure mathematics (i.e., first order logic, temporal logic, high order logic, object-calculus, sigma calculus, p-calculus, etc.) or ontologies [20]. The main goals of formalization are:

- To provide better understanding of design patterns and their composition; it helps to know when and how to use patterns properly in order to take advantage of them.
- To resolve issues regarding relationships between design patterns; it is not only important which design patterns are used to solve a given problem, but also it is important in which order they are applied.
- To allow for the development of tools that support activities regarding design patterns; these activities include semantic search for design patterns, automatic code generation and formal validation of design patterns.

1) Mathematics-based approaches: For example, Cornils and Hedin [21] described design patterns using reference attributed grammars with syntactic and context-sensitive rules. Eden et al. [22] derived LePlus (LanguagE for Patterns UniformSpecification) from Higher-Order logic to represent design patterns as logic formulas, which consist of the elements of object-oriented language (i.e. classes, methods or hierarchies) and relationships between them. Smith and Stotts [23] extended sigma calculus, which defines relationships between the elements of object-oriented language to describe design patterns.

Finally, Taibi and Ling Ngo [24] specified the structural and behavioral aspects of design patterns using first order and temporal logics, respectively. This approach described patterns in terms of classes, attributes, methods, objects and untyped values, and relationships between them.

 ISBN: 978-1-61208-158-8

The approaches adopted mathematics to describe design patterns. However, this can turn into a big disadvantage if users lack mathematical skills and thus, cannot easily understand how patterns have been described. Another big disadvantage with these approaches is that they describe the pattern solution only.

2) Ontology-based approaches: In recent years, several attempts have been made to formalize design patterns using ontologies, thus making the design patterns understandable for both humans and machines - this is the same idea as used in the scope of the Semantic Web[25]. For example, Rosengard and Ursu [15] introduced the idea of representing patterns as ontologies with a view to the development of tools for the automatic organization, retrieval and explanation of reusable solutions to software development, codes of good practice and company policies.

IV. Addressing a Larger Application Domain

Existing pattern languages are specialized on a closed problem domain allowing them to provide detailed knowledge for a certain class of design problems. However, we want to cover more than one aspect or problem domain and therefore search for ways to increase the extensibility of a pattern language. This also implies that not all formulated pattern can be chosen during the whole design process since readers have to decide for certain paths. Another point we want to address is the vividness of a pattern language. These new requirements were formulated in preliminary work on which we built in this work (cf. [2], [26]).

Analyzing related work, it can often be found that a group of authors defines a set of patterns from their experience and interlink them within their pattern language construction. However, in case that a pattern looses validity or is refuted by other readers who applied it, there is no process to feed back the experience into the pattern language. The approach resembles more a"try out and see" approach. Additionally, new ideas and found concept also cannot be included into the existing structure since there is no way for adding new nodes except for trying to include them together with the original authors in a new edition of the mostly written publication or by defining (yet) another custom pattern language. The new concepts and our proposed modification in the traditional pattern structure are described in the next chapter.

V. Introducing New Concepts

The pattern language approach described in this work extends the present approaches to functions that enable the construction of a collaborative community platform that we refer to as *Pattern Management System (PMS)*. With the help of this system, working application design approaches can be extended, modified and discussed.

In order to realize the PMS that also manages rules and processes for contributions, we decide for presenting the informal pattern language concepts (cf. III-A) to the reader. For the technical realization we need to consider at least semi-formal mechanisms to handle the pattern collection. In order to meet our requirements concerning a wide application domain and the mentioned community mechanisms, we extend the "traditional" pattern language concepts and pattern structures as described in the following sections.

A. New Concepts for a Pattern Language

The pattern language structure described in this work will follow the general concepts as described by [10] and [12] (cf. section III), but will also introduce new features resulting from the demand of covering a wider field of applications in the UbiComp domain as well as the support for external authors' contributions (cf. Figure 1 for illustrations).

Branching: While browsing the patterns, we introduce decision nodes at which the reader needs to decide for a subset of design recommendations. The decision may be based on interaction style, privacy demand or, e.g., device footprint. After making a decision, other subsets of design patterns are no longer available since the criteria for the intended design are not met any longer. This leads to a dynamic graph structure of causally related patterns.

Sequencing and Recommendation: Patterns that are located on the same level within the hierarchy represent design alternatives that can be chosen but also be implemented in parallel.

For the reader, it may be interesting which patterns where followed the most after deciding for certain predecessors and therefore providing a safe route of patterns to apply for a proven design pattern combination. Sequencing may be inferred from most read patterns or proposed combinations by others as well as on rankings. That way, different alternative paths can also be weighed by recommendations given by the community or by (semi-) automated proposals.

Community Involvement: The pattern language approaches discussed so far all relied on the knowledge of one

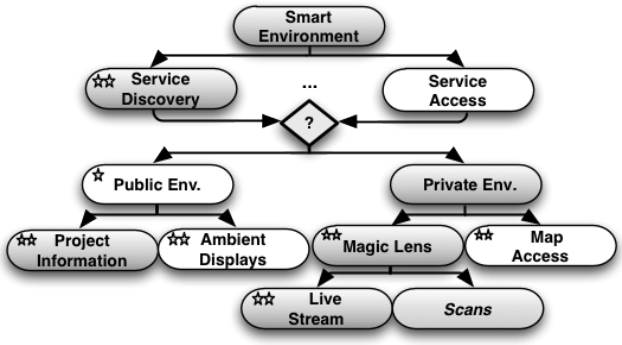

Figure 1. An excerpt of an example pattern language showing ratings as stars, relations as arrows, a necessary decision denoted by the question mark box, pattern sequences (gray) and one anti pattern node (*Scans*).

 ISBN: 978-1-61208-158-8

smaller group of authors and their, sincerely founded, domain knowledge. Additionally, writers workshops that help to formulate and refine patterns are organized during special Pattern Language of Patterns (PLoP) conferences [27]. Our approach addresses a larger amount of users especially application designers using the patterns and researchers familiar with the domain. The community will be given ways to comment on existing patterns and give suggestions for their refinement, provide additional references in literature or name counter-examples that confute the message of a design pattern.

Users can also propose new patterns to be linked within the pattern language structure. This way, the *vividness* of the pattern language is to be increased making it adoptable or extensible for more application domains within the UbiComp scope. Of course, rules and processes need to be defined in future work to guarantee controlled contributions.

Discussion and Refinement: Due to the fact that the patterns should not only be provided by one small group of authors and be ranked by them, rules and processes to propose new patterns or to discuss existing ones are needed.

This may result in a change of ranking, references to public work or parts of the description of the pattern. In case that only a smaller set of authors recommends and formulates patterns, the community feedback can also be interesting in order to see what concepts where really working and what needs to be changed in another iteration so that a pattern refinement is achieved.

Notes on the pattern hierarchy

The pattern hierarchy takes the dimension of time and implementation detail into account; the further the reader follows the structure, the more she learns about underlying concepts and design possibilities. At the lower levels of patterns, suggestions for implementations will follow.

B. Proposed Pattern Structure

According to the already existing pattern structures we describe the intended fields for an *UbiComp application design pattern* in the following. Some properties that were already defined by [9] and [10] were changed compared with the existing ones, new fields are marked by an asterisk(*).

Name: A pattern's name fulfills a vocabulary function by summarizing the intention of the pattern and the core of its solution. Finding a descriptive name which covers the pattern's intention and problem scope is not an easy task (cf. [10], p.65).

Ranking*: In the approaches described so far, patterns were ranked based on the experience of the author group providing the patterns eventually influenced by first reader groups. The ranking should be extended in such a way that readers and implementers of the pattern are enabled to provide feedback about the quality of the pattern. On the one hand with regards to readability and understandability, on the other hand regarding applicability, i.e., design success. That way, patterns can be refined or changed in an iterative way - also by third-party readers and consumers, respectively.

Illustration: Depending on the problem context, the media used for providing a quick glance at the problem should sensitize the reader and inform her quickly about the general design problem that is addressed by the pattern. The illustration may contribute to decision-making whether the pattern is relevant to the current situation.

Context(*): normally, the context is clearly defined by the pattern language structure. However, due to the restriction of design space given by decision nodes, readers have to keep track of taken decisions on the path to the particular pattern. It is possible that the problem domain is narrowed or changed to different problem cases by taking different decisions. This is only the case if nodes may have multiple parents.

Problem and Forces: In order to really capture the problem the current pattern is intended to solve, Borchers suggests to describe it as conflicting interests in a situation (opposing forces) that may result from physical facts but also may include social, economical and psychological constraints [9], [10]. These forces limit the design space or can even be contradictory such that they need to be discussed and eventually a compromise for a solution needs to be found. An example for a physical constraint is given by Alexander; Normally, a table in the center of a room pulls the visitors to that place. An opposing force is that people also like to stand close to large windows [9].

Example (Scenario): The inductive approach of providing examples helps novice readers to better understand the problem and solution described in a pattern. Secondly, experienced readers and professionals can already derive verifiable evidence for a working design solution. The example should be easy to understand and only cover the basic idea behind the pattern in order to provide a quick orientation to the readers.

Solution / Reason(*): This part provides a generally applicable design solution based on the lessons learned from the examples and literature references. It summarizes the central message of the design pattern telling the reader how to handle a specific design problem. In case that the pattern provide an anti-solution, this part turns into a description of the reasons for the failure of the concept. Both, the solution or the reasons should be described succinctly and therefore deliver the key message of the design pattern.

Diagram(optional): Depending on the current problem, an appropriate medium is to be chosen to illustrate and summarize the design solution. The diagram also servers as reading and understanding help and thus provides keeping an overview of the pattern during a quick scan of the pattern language. Additionally, the diagram is intended to assist in remembering the design solution.

Evidence in Literature*: Besides examples that already provide some evidence for a working solution, we regard

 ISBN: 978-1-61208-158-8

references in literature as very important for supporting the design recommendation inside a pattern. That way, application designers and researchers have another way of judging a design pattern besides the collaborative ranking mechanism described above. They can strengthen or weaken the design patterns statement by a custom literature research or custom evaluations that were published. Welie presents a similar concepts on his website where links to literature and a section for discussions are available [28].

Currently, design knowledge can be taken from literature and existing design pattern languages or best practices. In the large scope of UbiComp application design, the gap between knowledge in literature and (not yet) existing design patterns could be bridged.

As time passes, it could even happen that a pattern for design recommendation can be turned into an anti-pattern and vice versa, depending on evaluations, ranking and findings in application and research.

References*: These are pointers to other patterns of interest. In the pattern language approaches so far, the reader was encouraged to have a look at the referred patterns. The successors were mostly more detailed descriptions of a design concept based on size [9], time [10] or technical implementation level [12]. We extend the concept in three ways:

- By introducing *decisions*, only a subset of followers can be chosen by the readers. After choosing a subset of linked patterns, other successors may no longer be of interest for the pattern sequence.
- Additionally, the reader is supported by *most-read successors* based on the reading choice of others. This way, well-proven pattern combinations and sequences can be defined.
- It also has to be checked whether a certain reference is still valid. Earlier decisions may have cancelled out certain references. This is a big difference compared to the other pattern language approaches. Here, the hierarchy was mainly used for the grade of detail.

Decision-attribute*: If set, this new feature indicates a choice that needs to be made by the reader for the references leading to the current patterns successors. In order to capture a wide application domain, the pattern language needs ways to differentiate between alternatives. They concentrate on certain aspects of application design contexts, e.g., private vs. public information display or personal vs. shared devices.

Depending on the choice, a subset of following design patterns is available; others are cancelled out for that design property.

Anti-Pattern-attribute*: The intended pattern language approach also incorporates the explicit formulation of anti-patterns besides design patterns.

Marked as an anti-pattern the described design approach should be read as a DON'T rule for design. It is important that not every failed approach is documented by an anti-pattern. Thus, only surprising, non-trivial results should be documented supporting application designers that might have a similar idea to avoid the same design flaw.

Additionally, anti-patterns can also provide ideas for alternative concepts; once the application designer has followed a pattern sequence, she could also think of changing some parameters. An already formulated anti-pattern could help to avoid an already disproven idea and to try out yet another concept.

Counter-attribute*: The counter attribute is intended to keep track of pattern combinations indicating the strength of relation between the current pattern and its predecessors and its successors, respectively. The more frequent a pattern combination is selected by readers, the stronger the relation between them could be.

Accompanied by the ranking of each pattern, statements about pattern combinations and pattern sequences could be made.

VI. Summary

In this work, we addressed the requirements for establishing a pattern language structure for UbiComp application design as proposed in [2]. Based on existing concept from architectural, HCI and HCHI design patterns formulated by [9], [10], [12], we formulated new features for the existing pattern language structures.

The intention is to widen the application scope and to involve the community reading and working with given patterns and providing ways to express the feedback by rating and refining existing patterns or contributing new findings as patterns to be integrated into the pattern language.

In order to integrate patterns into the structure, the traditional pattern structure was extended by new features or existing features were partially refined to fit into the scope of the intended pattern language approach.

The extended pattern language concepts and pattern structure will result in a collaborative community platform, called *Pattern Management System (PMS)*, incorporating rules and processes for changing and refining existing patterns as well as ways to contribute new application design knowledge as patterns.

VII. Future Work

Future work will cover the following steps that build on top of each other:

Firstly, an initial set of UbiComp application patterns will be formulated and arranged in the pattern language.

Secondly, a first version of an online community portal to read and rate the existing patterns will be deployed.

Thirdly, rules and processes for rating, discussing, refining and contributing to the pattern language will be formulated and implemented. The intention is to support pattern authors willing to extend or modify the existing structure. The

 ISBN: 978-1-61208-158-8

mechanisms for the pattern discussion and refinement will be based on the concepts applied during writers workshop at Pattern-Language of Patterns (PLoP) conferences [27].

Currently, we also explore ways to implement the pattern and pattern language structure based on ontologies.

ACKNOWLEDGMENT

Irina Astrovas work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF). René Reiners' work was supported by the European Commission within the FP7-SECURITY project BRIDGE (no. 261817).

REFERENCES

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] R. Reiners, "Towards a Common Pattern Language for Ubicomp Application Design - A Classification Scheme for Ubiquitous Computing Environments -," in *Proceedings of the International Conferences on Pervasive Patterns and Applications (PATTERNS 2010)*, IARIA Conference. City, Portugal: Think Mind(TM) Digital Library, Nov. 2010, pp. 28–33.

[3] R. Ballagas, S. Kratz, and E. Yu, "REXplorer: A Mobile, Pervasive Spell- Casting Game for Tourists," *Architecture*, pp. 1–6, 2007.

[4] N. Henze, R. Reiners, X. Righetti, E. Rukzio, and S. Boll, "Services surround you," *The Visual Computer*, vol. 24, no. 7-9, pp. 847–855, Jul. 2008.

[5] R. Wetzel and I. Lindt, "The Magic Lens Box: Simplifying the Development of Mixed Reality Games," pp. 479–486, 2008.

[6] J. Schöning, M. Rohs, and S. Kratz, "Map Torchlight: A Mobile Augmented Reality Camera Projector Unit," *Information Systems*, 2009.

[7] R. Reiners, M. Jentsch, and C. Prause, "Interaction Metaphors for the Exploration of Ubiquitous Environments," in *Scenario*. Brussels, Belgium: International Conference "ICT that Makes the Difference", 2009.

[8] R. Reiners and V. N. Wibowo, "Prototyping an Extended Magic Lens Interface for Discovering Smart Objects in a Ubiquitous Environment," in *Proceedings of the IADIS International Conference Applied Computing 2009*, W. Hans and P. T. Isaías, Eds., IADIS. Rome: IADIS Press, Nov. 2009.

[9] C. Alexander, *A Pattern Language: Towns, Buildings, Construction*. New York, New York, USA: Oxford University Press, 1977.

[10] J. Borchers, *A Pattern Approach to Interaction Design*, 1st ed. John Wiley & Sons, 2001.

[11] J. Tidwell, *Designing Interfaces*, 1st ed. O'Reilly Media, 2005.

[12] T. Schümmer and S. Lukosch, *Patterns for Computer-Mediated Interaction*. Chistester, West Sussex, England: John Wiley & Sons, 2007.

[13] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*, 1st ed. Amsterdam: Addison-Wesley Longman, 1995.

[14] S. Montero, P. Díaz, and I. Aedo, *A Semantic Representation for Domain-Specific Patterns*. Springer-Verlag, 2005, pp. 129–140.

[15] J.-M. Rosengard and M. F. Ursu, *Ontological Representations of Software Patterns*. Springer Berlin / Heidelberg, 2004, vol. 3215, pp. 31–37.

[16] D.-K. Kim, R. France, S. Ghosh, and E. Song, "A UML-Based Metamodeling Language to Specify Design Patterns," in *Proceedings of the Workshop Software Model Eng. (WiSME) with Unified Modeling Language Conf. 2003*, 2003.

[17] G. Sunyé, A. L. Guennec, and J.-M. Jézéquel, "Design Pattern Application in UML," in *Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP '00)*. London, UK: Springer-Verlag}, 2000, pp. 44–62.

[18] M. Fontoura and C. Lucena, "Extending UML to Improve the Representation of Design Patterns," *Journal of Object-Oriented Programming*, vol. 13, pp. 12–19, 2001.

[19] (2011, July). [Online]. Available: http://uml.org

[20] L. Pavlič, M. Heričko, V. Podgorelec, and I. Rozman, "Improving Design Pattern Adoption with an Ontology-Based Repository," *Informatica*, vol. 33, pp. 189–197, 2009.

[21] A. Cornils and G. Hedin, "Tool support for design patterns based on reference attribute grammars," in *Proceedings of WAGA'00*, Ponte de Lima, Portugal, 2000.

[22] A. H. Eden, A. Yehudai, and J. Gil, "Precise Specification and Automatic Application of Design Patterns," in *12th IEEE International Conference on Automated Software Engineering (ASE'97) (formerly: KBSE)*, 1997.

[23] J. M. Smith, D. Stotts, and C. Hill, "Elemental Design Patterns : A Link Between Architecture and Object Semantics Elemental Design Patterns - A Link Between Architecture and Object Semantics," 2002.

[24] T. Taibi, "Formal Specification of Design Patterns - A Balanced Approach," *Journal of Object Technology*, vol. 2, no. 4, pp. 127–140, 2003.

[25] (2011, July). [Online]. Available: http://semanticweb.org/wiki/Main_Page

[26] R. Reiners, "An Extended Pattern Language Approach for UbiComp Application Design," in *Bonner Informatiktage*, 2011.

[27] (2011, July). [Online]. Available: http://www.hillside.net

[28] (2011, July). [Online]. Available: http://www.welie.com/patterns/

 ISBN: 978-1-61208-158-8