Towards a Common Pattern Language for Ubicomp Application Design - A Classification Scheme for Ubiquitous Computing Environments -

René Reiners
Fraunhofer FIT
Schloss Birlinghoven
53754 Sankt Augustin
rene.reiners@fit.fraunhofer.de

***Abstract*—The idea of *Ubiquitous Computing* was first formalized and described by Mark Weiser in the early 90's. Since then, it has been followed by many research groups and extended in many ways. There are many ideas, concepts, prototypes and products implementing ubiquitous computing scenarios. However, the manifold of approaches also brings along a large variety of denominations for eventually similar concepts. Our work seeks for the creation of a dynamic pattern language gathering design knowledge for ubiquitous computing applications and the underlying concepts. The intention is to support researchers and application designers in the domain to avoid the repetition of design errors and provide design knowledge about successful approaches. In order to get closer to that aim, our first step is to present a classification scheme applicable to existing and future approaches that is needed in order to collect, structure and compare application design approaches as design patterns.**

Keywords- ubiquitous computing; pervasive computing; application design; classification; pattern language

I. Introduction

The research field of *ubiquitous computing*, also referred to as "ubicomp", was founded by Mark Weiser who presented the concept's idea in his work "The Computer of the 21st Century" [1]. Smart devices that are equipped with sensors or that are capable of providing information silently integrate into the environment. The communication between different entities should ideally take place in a wireless manner. This way, a number of smart devices together shape a *ubiquitous computing environment*.

The current situation where a personal computer drags all the attention towards itself should completely be avoided such that users are able to concentrate on the tasks they wish to perform instead of caring of the interaction. Computations can be performed inside the smart devices themselves or performed on machines inside the room. Connection and tasks management must not be the user's concern.

Weiser compares his idea to the ancient art of writing. Nowadays, we consume and provide information by simply reading or writing it - we are making use of this technique although we do not mandatorily need to know how to produce ink or paper, for example.

The concept of working with technology without having to know much about the details of the underlying infrastructure is the core idea of ubiquitous computing. Working also means using or even living in ubiquitous computing environments.

Currently, devices that can be used in a very "ubiquitous" way are entering the market; mainly these are netbooks and smartphones. This class of devices are first candidates to make ubiquitous computing widely available since there is a still growing increase in sales numbers as stated by Gartner [2]. With a high degree of connectivity and new generations of different kinds of sensors, new applications and ways of interaction become possible that were still visionary some years ago.

A. Functionality Everywhere

When talking about different service networks and the provisioning of services, there is also the need for looking beyond the personal (and limited) scope of mobile devices. Following the concepts of *Pervasive Computing*, the *Internet of Things (IoT)* or *Cloud Computing*, there are far more possibilities to offer services, since:

- The concept of *Pervasive Computing* allows the integration of computing power into real world objects, devices and environments [3].
- *Labeling* and therewith the IoT concept gives the possibility to uniquely identify and address real world objects [4]. Thus, the possibility of potentially unlimited labeling holds chances and challenges for many different kinds of applications as outlined by [5].
- In case that resources are too weak or cannot fulfill the requirements of task, these tasks are outsourced into the *Cloud* and thus virtually extend the devices' resources and transform them into gateways accessing more powerful functionalities [6].
- The growing *network infrastructure* allows communication between devices and thus the exchange of information or the consumption of services. An overview of the mobile phones and network infrastructure generations can be found online at [7].

Services that are deployed together with real world objects and which are accessible via any kind of network are called *smart services* for the rest of this document.

B. Realizations

The above concepts and visions are partially already a reality. Amazon or Google, for example, provide access to their processing powers by introducing the Amazon Elastic Compute Cloud [8] or the Google App Engine [9].

Additionally, a combination of using GPS data together with permanent network access by mobile providers are used in projects like Layars [10] or Wikitude [11]. These mobile approaches make use of the mobile device's position and access databases over the mobile network in order to present additional information about objects next to the current position. These approaches are based on the Magic Lens approach first introduced by Bier et al. [12].

Later work incorporates mobile projectors to augment physical objects like paper maps and project information directly onto the object [13].

Further ideas than only receiving information are applicable which is for example already realized by the UbiLens project at Fraunhofer FIT [14]. In this project, smart services attached to different kinds of real world objects can be consumed, ranging from information retrieval over triggering actions up to the combination of different services and devices. An online available example shows how different real world objects are recognized by a server in the background receiving the camera image from the mobile phone. After identification of the object, different services can be selected on the mobile phone according to the purpose of the real world object [15].

II. Problem and Approach

A manifold of research approaches explore ways to offe ubiquitous functionality in public and private environments. Since each approach concentrates on different scenarios, uses different hardware and calls its components differently, it is hard to find similarities within existing implementations.

We see the danger of the repetition of design failures, unused chances of extending successful designs and missed chances to learn from realized approaches. These dangers may result in loss of design time and even money.

We consider a dynamic pattern language extracting and structuring results from different approaches as a possible solution to that problem. The pattern language will help new application designers in the field to more easily find working design approaches and modify them to their needs. This will also support the knowledge management within research communities and enterprises. Domain novices can pick up expert knowledge gathered from experience and formulated in the pattern language.

The rest of the paper is structured as follows: Section III provides an overview of different pattern languages used in different domains. In the preceding section (cf. section IV), we discuss features that are, from our point of view, missing in the current approaches of pattern languages. In section V, the idea of the common classification scheme is shaped addressing one of the problems discussed. The last section gives an overview of the planned next steps towards the intended pattern language and its intended new features (cf. section VI).

III. Pattern Languages

There is a variety of application domains and the popularity of gathering knowledge in pattern languages that are more or less technically formulated.

In the domain of HCI design patterns, Borchers, Schümmer and Stephan Lukosch follow the basic structure of Alexandrian design patterns (cf. [16]) by making use of natural language in order to describe solutions to specific design problems [17], [18]. In their approach, they structure a pattern into the following parts which are often similarly adapted in other pattern languages:

A *name*, *sensitizing picture*, the *intent* summarizing the pattern's solution in one sentence, the *context* in which the problem occurs and the solution is described, a *problem* description containing the most important aspects, a *scenario* putting the pattern's problem into an illustrating example context in order to increase understandability, *symptoms* helping the reading to find out about conflicting forces within the context, a *solution* to the conflicting forces problem, *dynamics* naming actors and components involved in the pattern, *rationale* providing explanations for a pattern's success and applicability, *checks* that pose questions that try to help the reader to figure out whether the pattern representing a template solution was well adapted to the current design problem, *danger spots* showing potential new problems that may occur when applying the pattern.

So they can be regarded as warning features trying to avoid the blind application of a pattern. The closing sections of a pattern are named *known uses* representing the second part of a pattern's "proof" by presenting approaches in which the pattern is successfully applied and *related patterns* linking to relevant alternatives, patterns that are important for other stakeholders or patterns that go into more detail of a possible solution.

Pattern languages are also to be found in different application domains reaching from technical software design in object-oriented programming (cf. [19]) to interface design up to organizational patterns in business structures.

Rising and Manns, for example, present ways to restructure existing organizational structures and to introduce new ideas into an existing system. Their pattern language *Fearless Change* relies on social structures and requires practices that establish trust in new goals [20].

The *Organizational Patterns* language concentrates on team interaction in software projects as described by [21].

Another example is the *TELL* project that present patterns for computer-supported learning [22].

Teachers are supported in questions about group-based learning with collaboration technology find advice in the *Pedagogical Patterns Project* as described by [23] and in the domain of business process management, the approach by [24] represents an attempt to describe socio-technical systems following strict business processes. The pattern language developed is called *Workflow Patterns*.

The domain of user interface design is also served by the pattern languages *Web Usability Features* ([25]), *Web Patterns* [26] and for example the *Amsterdam Pattern Collection* [27]. Tidwell presents a very comprehensive pattern language for computer-mediated interaction in the context of non-web-based applications [28].

In many different fields where knowledge and experience can be captured interlinked, pattern languages have proven to be a useful approach.

IV. Missing Features

Current pattern language approaches provide a good structural basis for capturing design knowledge for specific application domains mostly explored by a small group. However, concerning the idea of covering design guidelines from many different projects and groups, we consider more features for a pattern language extending the current concepts as needed. These features are described in the following.

A. Lacking Application Domain Independence

Current pattern language approaches are mostly bound to one specific application domain and a small set of involved group. From the given circumstances and denominations, patterns are created and arranged in a pattern language.

The intended pattern language is intended to cover many different research groups and commercial projects, where different ideas and approaches are implemented and evaluated.

Many approaches are situated in the field of ubiquitous computing but the *naming* of components, techniques and concepts differs widely. A comparable classification and naming scheme is needed in order to be able to generalize concepts and interconnect common knowledge as well as to search within existing work results.

This is, in our opinion, a needed requirement to be able to integrate different approaches in different application domains.

B. Lacking Extensibility and Openness of Knowledge

Even after extracting knowledge about a certain aspect of ubiquitous computing interaction design, it is hard to discuss results from existing and new approaches. Once published, they remain within the documents and need to be refined or discussed besides the actual publication in follow-up research, forums or conferences.

That makes the reuse and refinement of results a very hard task. At the moment, there are only limited ways to extend and discuss research results. Openness is only given in a passive way; the results can be read but not actively be extended or discussed.

C. Lacking of Recommendation

Considering patterns about applied concepts and techniques within the domain, there is no direct connection between them. The exchange of knowledge about combinations that were implemented and worked out well and those which did not is not given. Recommendations for proven combinations of concepts are missing and therefore hindering the reuse and extension of concepts.

Again, time-to-market and time-to-research-results can be shortened by providing suggestions for good combinations of smaller units of solutions from different approaches.

D. Lacking of Knowledge about ”Bad Practices”

In publications, often results reveal information about working concepts that were successfully implemented. Only initial studies about a certain problem domain concretely outline deficits in order to justify and motivate intended research.

However, in a domain that is actively being explored, failures or methods that were not accepted are not always clearly described or even mentioned. In our opinion, this is often the case in research about interaction techniques and metaphors. Here, authors mainly describe working solutions and drawbacks are omitted.

We consider the inclusion of bad practices that are not trivial and were revealed unexpectedly in experiments and implementation a very important feature to be integrated into the pattern structure.

V. A Classification Scheme for Ubiquitous Computing Environments

As a first step towards a pattern language of application designs, a common denomination for similar approaches is needed. Once different approaches can be described by a common vocabulary, the inherent design knowledge can be compared more easily, discussed and transferred to different application scenarios.

In ubiquitous computing environments, there are different objects providing functionality and interoperating among each other, the environmental infrastructure and the user. Different kinds of functionality is identified and abstracted to a semantic level therefore called *smart service* (cf. Section V-B). Real world objects that are augmented with smart services are referred to as *smart objects* (cf. Section V-A). Smart objects augmented with an arbitrary number of smart services together build a *smart environment* within an application domain (cf. Section V-D) .

The definitions given in this work will constitute the foundation of the intended pattern language approach and cover already existing applications that are found within current research and projects but will also be use to describe future ideas and approaches.

A. Smart Objects

In the scope of this work, a *smart object* can be any kind of device of object that provides *functionalities* augmenting its original purpose. This could be the provision of information related to the object or something more abstract it stands for. For example, a standard paper timetable in train stations could provide the same information that is printed on it in a virtual way. Furthermore, it could also stand for advanced routing in such a way that travelers use it to decide for a route from the current station to their destination. This routing could also be offered as a computer-supported services like it can be found on current websites driven by public transportation companies.

The idea is now to be able to attach any kind of service to a standard object or device which ideally are related to the object's original purpose or more abstract concepts it stands for.

Thus, in the scope of this work, any kind of object potentially provides an arbitrary number of smart services that provide information or extended functionalities. Consequently, every object that is augmented this way is referred to as a smart object.

Fig. 1 illustrates this concept showing arbitrary functionality, i.e., smart services, being connected to a real world object.

B. Smart Services

Smart services are virtually attached to physical objects or devices and therefore augmenting them with virtual functionalities. The functionality they provide could be informative or offer more sophisticated applications. Ideally, the services are related to the object's original purpose but theoretically, they could offer anything developers have in mind.

However, the applicability and user's understanding would definitely suffer from such implementations since they would not really convey a coherent meaning like in the timetable example given in Section V-A.

Currently, the derived classification scheme covers three different kind of smart services. They can

- provide information (*provider*)
- provide ways of interaction (*connector*)
- process input (*consumer*)

Each kind of smart service optionally possesses a special attribute which is called the *takeaway - attribute*. This optional specialization of a service enables the user to take the offered functionality with her so that it still available later and when she is not necessarily close to the smart object. A service which does not posses the takeaway-attribute can only be used within direct contact with the smart object.

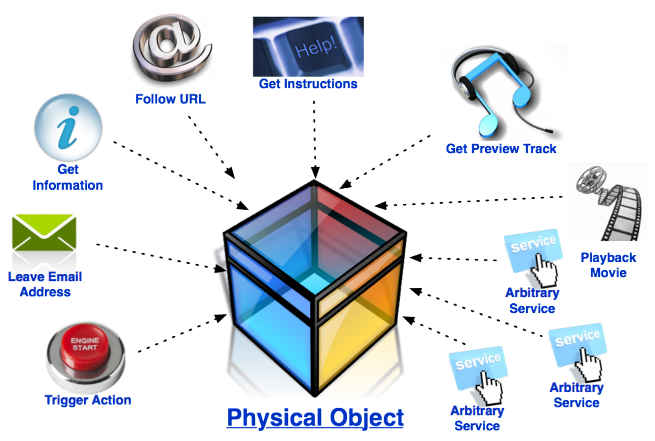

Figure 1. A standard real world object can be augmented by an arbitrary number of different kinds of smart services. The functionality can be freely defined.

In order to illustrate the classification scheme, one small example per kind of service is given in the following. Additionally, a possible implementation and usage of the takeaway-attribute is explained.

(i) *Providers* are implemented as services that offer certain information about a special location like point of interest (POIs). The Wikitude project is one example of an application that provides information bound to certain buildings or locations. Google Goggles (cf. [29]) also applies this approach by connecting information to buildings.

(ii) *Connectors* enable the user to control an offered service. One example is a public display, that offers a smart service allowing users to remotely control the display. That way, the user's input is processed and directly fed into an application lying behind the service. Also, the timetable scenario mentioned in Section V-A can be implemented in an interactive way such that the user influences parameters like price, travel duration or comfort class when planning his connection.

(iii) *Consumer services* do not necessarily provide sophisticated feedback to the user or provide a direct result. They are moreover be regarded as triggers for certain processes or applications. Examples for this are push-services that for examples upload just-taken pictures to an online community account like Facebook.

Another example is represented by macros like those used in the home automation sector. A service called ”turn on the lights” is then used as a trigger. This way, the user's input is consumed and a whole system, the home automation system in this case, takes over.

C. The Optional Takeaway-Attribute

As an optional add-on, the takeaway attribute comes into play. In the context of the timetable example *providing* information about connections, this means that the train arrival and departure times can be taken away by the user and therefore are also be available when she is not standing next to the timetable. That way, information can be collected and consumed later and repeatedly.

For *connectors*, it makes sense to use an application connected to an interactive service also at a later time. The can make a train booking at home but have the virtual service still available for tracking the booking process or eventual changes. So users are able to connect a real world object to a *virtual pendant* offering similar or extended functionalities.

Consuming services can also offer the takeaway-attribute. In the example of home automation, the user is then able to turn on the lights while she is still outside and does not first have to search for the service in the dark.

Fig. 2 shows some examples of service-augmented real world objects, optionally with the take-away attribute. In the example, an ordinary timetable at a train station is augmented with service finding the train connection from the current location to the desired destination. This service posses the takeaway-attribute since it could also be used again at a different location even if the user is not near the real world object.

The TV screen as an ambient display offers the service to provide video content that can be played on the screen but also played back on te the user's mobile device. Another service combined with ambient speakers offered is the playback of user-provided content that is only available when she is present at the screen. Remote control is not allowed here.

In the case of the coffee-machine, the takeaway attribute is not given since this services of filling a cup or retrieving information about the coffee status is only available when a user is directly interacting with it due to security reasons.

These examples show that the takeaway-attribute needs to be used carefully in order to provide meaningful services. Services that provide physical feedback like printouts (e.g., tickets) or products (e.g., coffee from an augmented coffee machine) need to ensure that the user is directly available.

D. Smart Environments

A setup with arbitrary kinds of services attached to an arbitrary number of real world objects, is referred to as *smart environment*. Different classes of smart environments are possible. Based on the purpose of the services (entertainment, technical support, maintenance) or the location of the environment and the level of publicity or privacy, respectively. Examples for these kinds of environments are public spaces like train stations, official buildings, airports or sights in a city. More private or security related situation can be found at home or in office spaces.

Applications like Wikitude (cf. [11]) or Layars (cf. [10]) already turn public spaces into a kind of "informational smart environment" by displaying additional information about a location the user is close to.

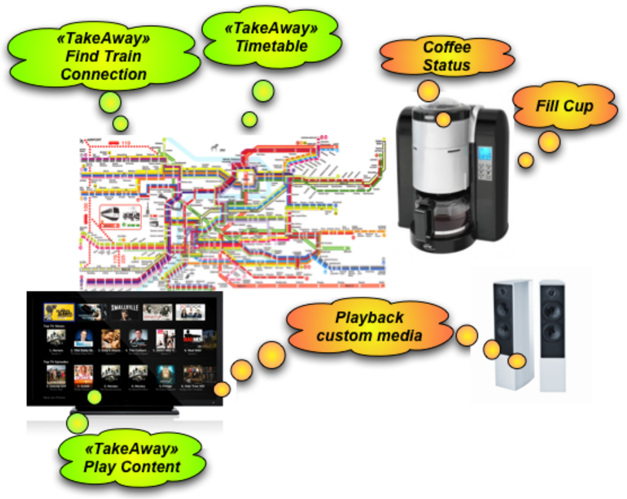

Figure 2. In this example, services with the takeaway-attribute are connected to a metro plan and a TV screens. The coffee-machine and speakers providing playback services but only allow direct interaction.

VI. Future Work

The long-term intention of our work is to develop a pattern language for application design in ubiquitous computing environments. As a first requirement for the creation and extension of such a structure we consider a *common vocabulary* as a necessary requirement for comparing different approaches and to extract knowledge from them. The extraction will be presented in form of design patterns, similar to approaches presented by Schümmer and Lukosch, Borchers and Gamma et al. (cf. [18][17][19]). The latter work addresses the technical part of application design in terms of how a technical problem can be solved by applying software patterns. The former patterns are formulated against specific application domains, i.e., CSCW and HCI, that primarily describe design knowledge of applications on a conceptual level. Technical suggestions play a minor role. Like Borchers, Schümmer and Lukosch, the patterns for ubicomp application design are intended to be arranged in a pattern language and therefore interconnecting patterns. The deeper readers follow the structure, the more details of the application design are described.

New features like decision nodes separate different kinds of interactions depending on the usage scenario. Recommendation mechanisms will help to find successful combinations of patterns. Finally, the pattern language is intended to be open to new patterns that can be integrated.

The design patterns will range from the discovery of smart services, over the interaction until user preferences and privacy and security patterns.

VII. Conclusion

This work presents first conceptual steps in the progress of finding a classification scheme that is able to map denominations from existing and future approaches to a generally applicable vocabulary needed for the formulation of design patterns in the domain of ubicomp application design. Next, existing approaches in the domain of mobile applications in ubicomp environments will be analyzed and the presented scheme will be mapped to these approaches. After translating the approaches to the abstract vocabulary, patterns of successfully implemented concepts supported by published evaluations will be derived and discussed within the community. From the initial set of patterns, the other requirements described in section (II) will be addressed.

In a later step, new domains and ideally a large set of domains are to be analyzed. The results will either support the generality of the developed pattern language and its new features or confute the approach. From our point of view, the assembly of patterns in a pattern language will support the gathering, structuring and extraction of design knowledge from current and future approaches and make them comparable and thus facilitate discussion, reuse and modification throughout application scenarios.

REFERENCES

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

[2] "Gartner newsroom - press release, may 19th, 2010," 2010, http://www.gartner.com/it/page.jsp?id=1372013.

[3] U. Hansmann, L. Merk, and M. S. Nicklous, *Pervasive Computing - The Mobile World*. Berlin: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2001.

[4] H. Chaouchi, *The Internet of Things: Connecting Objects*. ohn Wiley & Sons, 2010.

[5] C. Floerkemeier, M. Langheinrich, E. Fleisch, and F. Mattern, *The Internet of Things: First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008, Proceedings*, 1st ed. Springer-Verlag Gmbh, 2008.

[6] J. Rhoton, *Cloud Computing Explained: Implementation Handbook for Enterprises*, 2nd ed. Recursive Press, 2010.

[7] C. M. S. Ltd., "Telecoms market research," website, 2008, http://www.telecomsmarketresearch.com/resources/Mobile_Phone_Market.shtml.

[8] "Amazon elastic compute cloud," 2010, http://aws.amazon.com/ec2.

[9] "Google app engine," 2010, http://code.google.com/intl/appengine/appengine.

[10] "The layars project," 2010, http://layars.com.

[11] "The wikitude project," 2010, http://wikitude.org.

[12] E. A. Bier, K. Fishkin, K. Pier, and M. C. Stone, "Toolglass and magic lenses: the seethrough interface," *Proceedings of SIGGRAPH*, vol. 93pp, pp. 73–80, 1993.

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

[14] V. N. Wibowo, "The UbiLens Approach - Visualisation of and Interaction with Real World Objects through a Moble Phone's Camera ," master thesis, Fraunhofer FIT, 2010.

[15] "Gartner newsroom - press release may 19th, 2010," 2010, http://www.youtube.com/watch?v=IY1FmKhfAao.

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

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

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

[19] 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.

[20] L. Rising and M. L. Manns, *Fearless Change: Patterns for Introducing New Ideas: Introducing Patterns into Organizations*, 2005th ed. Amsterdam: Addison-Wesley Longman, 2005.

[21] J. O. Coplien and N. B. Harrison, *Organizational Patterns of Agile Software Development*, illustrated ed. Prentice Hall International, 2004.

[22] F. L. National Centre and Literacy, "The Tell Project." [Online]. Available: http://www.tell.praesa.org/

[23] J. Eckstein and J. Bergin, "The Pedagogical Patterns Project," 1999. [Online]. Available: http://www.pedagogicalpatterns.org/

[24] C. Hentrich, "Six patterns for process-driven architectures," in *Proceedings of the 9th Conference on Pattern Languages of Programs (EuroPLoP 2004)*, 2004.

[25] I. Graham, *A Pattern Language for Web Usability*. Amsterdam: Addison-Wesley Longman, 2003.

[26] D. Schwabe and G. Rossi, "The object-oriented hypermedia design model," *Communications of the ACM*, vol. 38, no. 8, pp. 45–46, August 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=208344.208354

[27] M. van Welie, "The Amsterdam Pattern Collection," 2010. [Online]. Available: http://visiblearea.com/cgi-bin/twiki/view/Patterns/Amsterdam_Collection_of_Interaction_Design_Patterns

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

[29] "Google goggles," 2010, http://googlegoggles.com.