Please quote as: Knote, R.; Baraki, H.; Söllner, M.; Geihs, K. & Leimeister, J. M. (2016): From Requirement to Design Patterns for Ubiquitous Computing Applications. In: 21st European Conference on Pattern Languages of Programs (EuroPlop '16), Kaufbeuren, Germany.

From Requirement to Design Patterns for Ubiquitous Computing Applications

ROBIN KNOTE, HARUN BARAKI, MATTHIAS SÖLLNER, KURT GEIHS, JAN MARCO LEIMEISTER, Research Center for Information System Design (ITeG), University of Kassel, Germany

Ubiquitous Computing describes a concept where computing appears around us at any time and any location. Respective systems rely on context-sensitivity and adaptability. This means that they constantly collect data of the user and his context to adapt its functionalities to certain situations. Hence, the development of Ubiquitous Computing systems is not only a technical issue and must be considered from a privacy, legal and usability perspective, too. This indicates a need for several experts from different disciplines to participate in the development process, mentioning requirements and evaluating design alternatives. In order to capture the knowledge of these interdisciplinary teams to make it reusable for similar problems, a pattern logic can be applied. In the early phase of a development project, requirement patterns are used to describe recurring requirements for similar problems, whereas in a more advanced development phase, design patterns are deployed to find a suitable design for recurring requirements. However, existing literature does not give sufficient insights on how both concepts are related and how the process of deriving design patterns from requirements (patterns) appears in practice. In our work, we give insights on how trust-related requirements for Ubiquitous Computing applications evolve to interdisciplinary design patterns. We elaborate on a six-step process using an example requirement pattern. With this contribution, we shed light on the relation of interdisciplinary requirement and design patterns and provide experienced practitioners and scholars regarding UC application development a way for systematic and effective pattern utilization.

• Software and its engineering~Patterns • Software and its engineering~Design patterns *• Software and its engineering~Requirements analysis • Software and its engineering~Software development methods • Human-centered computing~Ubiquitous and mobile computing • Human-centered computing~Ubiquitous and mobile computing theory, concepts and paradigms • Human-centered computing~Ubiquitous computing*

Additional Key Words and Phrases: Requirements Engineering, Requirement Patterns

ACM Reference Format:

Knote, R., Baraki, H., Söllner, M., Geihs, K., Leimeister, J. M. From Requirement to Design Patterns for Ubiquitous Computing Applications. In *21st European Conference on Pattern Languages of Programming (EuroPLoP'16),* Kaufbeuren, Germany, 2016

DOI: http://dx.doi.org/10.1145/3011784.3011812

1. INTRODUCTION

The concept of Ubiquitous Computing (UC) implies that computing is everywhere around us (Weiser, 1991) and that applications make use of sensor data and personal data to adapt autonomously due to context changes and personal preferences and profiles. The data may be processed on servers or devices that are not visible to the user and that may be accessible to third parties. The UC application might execute actions the user is not aware of and perhaps would not even allow. While this tight interweaving into the users' everyday lives offers a wide range of exciting application opportunities, it also demands the consideration of non-technical, social aspects. In this regard, social compatibility has to be kept in mind throughout the entire development process. That means that UC applications have to adhere to

Author's address: R. Knote, H. Baraki, M. Söllner, K. Geihs, J.M. Leimeister, Research Center for Information System Design (ITeG), Pfannkuchstr. 1, 34121 Kassel; E-Mail: {robin.knote; soellner; leimeister; geihs}@uni-kassel.de; baraki@vs.uni-kassel.de

EuroPLoP '16, July 06 - 10, 2016, Kaufbeuren, Germany

ACM 978-1-4503-4074-8/16/07…$15.00
DOI: http://dx.doi.org/10.1145/3011784.3011812

laws and social norms and factor in the users' perception and trust while at the same time providing functionalities that are characteristic for UC. Hence, questions of privacy, trust, usability, legal compliance etc. have to be addressed during system development. To meet this challenge and to provide proper solutions, software experts have to collaborate with experts from other disciplines.
Since stakeholders from many different disciplines work together towards a common goal, a special focus has to be laid on eliciting, specifying and documenting requirements. These requirements are based on normative, social and technical aspects and must be transferred into functional requirements that can be used for system development. As many real-world problems (e.g., related to trust or law) are similar for many UC application development projects, using patterns of standardized requirements seems to be a suitable solution. This consideration does not only apply to requirements engineering, but is also valid for addressing recurring requirements in the software design process using interdisciplinary design patterns. However, literature about integrating both requirements and design patterns is sparse and thus their relation in practical contexts has not been highlighted sufficiently.

In this paper, we introduce our field-tested approach of combining both concepts. We present a process for deriving design patterns from normative requirements in the spotlight of UC applications, focusing on social acceptability and considering both privacy and usability issues. In section 2, we will give an overview of the background and our motivation grounded in the VENUS project. Section 3 discusses related work to lay the foundation for our approach. The requirements and requirement patterns that we use in this paper to demonstrate our process are presented in section 4. In section 5, we elaborate on the process of finding suitable design patterns for the requirements which is our core contribution. We conclude with giving a short summary and highlighting future perspectives in section 6.

2. BACKGROUND

In the VENUS project (Geihs *et al.*, 2014), we have worked out a comprehensive interdisciplinary development methodology for the design of socially aware UC applications. The disciplines represented in VENUS were computer science, jurisprudence, ergonomics and trust research. The created methodology supports interdisciplinary teams to work together in an efficient and structured manner. The core of the methodology is described in the following sections. The detailed approach can be found in (David *et al.*, 2014).

Briefly, the VENUS methodology proposes to formulate initially the functional requirements in a technical but abstract way so that they do not provide any concrete technical solutions and such that they are understandable for experts from different areas. To this end, potential users and the aforementioned domain experts should be involved. The next step is to determine normative requirements that can be derived from laws, norms, standards and other provisions relevant to the application. They have to be translated to functional requirements with the aid of the respective domain experts. Hence, the VENUS methodology is an iterative development approach consisting of the phases: analysis of needs, requirements management, conceptual design, software design with implementation and in-situ evaluation (Hoffmann and Niemczyk, 2014).

VENUS manifests the 'big picture' in which the approach described in section 5 of this paper was used to find suitable design patterns from elicited requirements and, thus, is applied to integrate the phases 'requirements management' and 'conceptual design'. Hence, the process described in this paper is an integral part of the VENUS methodology which is of particular interest since it has not been specified in detail within related publications yet. We are particularly interested in the requirements management and elicitation phase as it is where requirements from various disciplines are elaborated and merged. Besides usual functional and quality requirements, requirements for legal compatibility, usability and trustworthiness are considered. Domain experts derive these requirements from normative sources by concretizing the provisions with regard to the technical system. The goal is to translate these provisions to technical requirements. This is done in several steps by the respective domain experts. The detailed procedure can be found in (Hoffmann and Niemczyk, 2014).

Although the VENUS development method fosters the interdisciplinary teamwork and supports the different disciplines to work out proper socio-technical solutions, we experienced that developing such applications remains a time-consuming and complex task. For that reason, in previous work we have analyzed three demonstrators (i.e., prototype applications that we have developed within our research project and evaluated to show whether and how the VENUS methodology works) implemented during the VENUS project and searched for recurring design patterns in the demonstrators that were related to interdisciplinary concerns about how to achieve certain aspects of the social embedding of the applications. The demonstrators have been designed and implemented by separate development teams in different application domains, i.e., Meet-U in mobile computing (Göschka and Haridi, 2012), Connect-U in social networking (Atzmüller *et al.*, n.d.), and Support-U in ambient assisted living (Hoberg *et al.*, 2012). Domain experts helped us in an iterative process to consolidate the list of extracted patterns and to pick out those that are applicable for common UC applications. The collection of patterns created is presented in (Baraki *et al.*, 2015).

However, these design patterns address implementation aspects but do not incorporate the phases of requirements engineering. Hence, we created and extracted requirement patterns (Hoffmann, 2014) by analyzing the demonstrators regarding the requirement elicitation phase. Requirement patterns are an approach to reuse recurring requirements (Wieringa and Persson, 2010) and to identify and document software requirements (Robertson and Robertson, 2006). Requirement patterns are applied for eliciting and analyzing requirements and can be considered a collection of knowledge and experience, which can be reused in development projects by adaptation (Wahono, 2002). Requirement patterns contain templates to describe a standardized requirement and other relevant information in tabular form (Toro *et al.*, 1999). These are, for example, the goal of the standardized requirement and relations to other patterns. To ease adaptation, attributes can be defined with usable content. However, only content that already has been elaborated and tested carefully should be predefined.

In this work, we try to connect both types of patterns, i.e., interdisciplinary requirement and design patterns, to establish a clearer structure for their utilization. The application of development methods, such as our VENUS method, will be simplified and accelerated by this kind of pattern languages since they are offering guidance over long phases in the development process and over several disciplines.

3. RELATED WORK

Presently, there is only limited work addressing interdisciplinary design patterns in the field of Ubiquitous Computing. Many works rather focus on enabling adaptivity or context-awareness but do not consider crosscutting aspects like transparency, trust, privacy and informational self-determination. The following section is listing primarily related work that adopts a clear interdisciplinary viewpoint.

Lahlou and Jegou (2004) propose nine guidelines, called European Disappearing Computer Privacy Design Guidelines. Most of them, i.e., Think Before Doing, Good Privacy Is Not Enough and Re-visit Classic Solutions, give some clues which thoughts developers should take up in general. Others are geared to the principles introduced by the OECD (OECD *et al.*, 2003). Altogether, Lahlou et al. indicate problems and challenges developers have to tackle. Many of these guidelines can be considered indeed as motivation for requirement and design patterns, but not as intelligible instructions developers and requirement engineers can apply.

One of the first publications looking at privacy and ubiquitous computing together is proposed by Langheinrich (2001). He introduces several fundamental principles such as the principle of notice in which users have to be informed by announcements first and foremost if data is collected about them.

Further principles like the choice and consent principle were justified on the basis of the EU Data Protection Directive 95/46/EC and the fair information practice principles (FIPP) that were first described in the Privacy Act of 1974 (5 U.S.C. § 552a as amended). Langheinrich considers them from the perspective of Ubiquitous Computing and suggests approaches of a general manner. His work does not imply common solutions that support their concrete implementation or formulation.

Chung *et al.* (2004) introduce 45 design patterns for ubiquitous systems, grouped into the four groups Ubiquitous Computing Genres, Physical-Virtual Spaces, Developing Successful Privacy, and Designing Fluid Interactions. Their intention is to provide an initial list of design patterns that can be enhanced and extended. In the context of this work, especially the fifteen design patterns devoted to privacy and the eleven patterns related to fluid interactions are of interest. Their very comprehensive set of design patterns did not emerge from the authors' own software projects but from studying the relevant related literature. For example, their Fair Information Practices pattern lists the aforementioned practices of the 1970s as solution and refers to Langheinrich's work (Langheinrich, 2001). Other patterns reproduce principles like the choice and consent principle, but do not provide clear instructions how to put them into practice. In contrast to our approach, Chung et al. consider privacy and usability separately. In our work, we take requirements from different disciplines at the same time into account since they can oppose each other and thus have to be coordinated.

Ruiz-López *et al.* (2013) propose various patterns to address non-functional requirements in ubiquitous computing systems. Most of them refer to adaptivity, reliability and security, but two of them, i.e., the Pseudonymity and the Human Factor patterns, are classified as patterns concerning ethics. The Human Factor pattern, for example, is a hybrid approach that allows the user to perform certain activities on his own instead of leaving it up to the software. According to this pattern, developers should consider the possibility to only assist users or to let users do things on their own to improve, inter alia, their well-being. Here again, detailed information about the pattern and possible scenarios and examples are missing. In fact, Ruiz-Lopez et al. present an excellent analysis, but their conclusions are formulated on a very abstract level and without concrete connections to interdisciplinary design guidelines.

Other works that consider interdisciplinary patterns from a more general viewpoint cannot be applied on UC systems without further considerations. Most of them focus on pure Internet applications or on traditional application areas (Hafiz, 2013). The characteristics of UC systems, especially their restrictions and their differentiating capabilities, which encompass context-awareness and adaptivity, necessitate adapted or new design patterns.

Considering the principles, guidelines and the patterns introduced in this section, two main problems can be identified. Firstly, the descriptions are too vague or they concentrate on pure technical challenges like adaptation and context-awareness, and thus are too detailed. An approach is required that provides support from the initial requirements through to their realization. Our approach is assembling interdisciplinary design patterns and requirement patterns tailored to UC applications.

4. REQUIREMENT PATTERNS

Since we aim at deriving patterns for recurring requirements, we follow Landay and Borriello (2003) who argue that requirements and designs are 'recurring' if they can be found in at least three good implementations. Considering our demonstrators Meet-U, Connect-U and Support-U, those requirements and design guidelines that are used for all three of them can be described in a pattern format. Thus, we identified nine requirements that have been equally considered and are valid for all demonstrators. These requirements, however, address system properties to improve the trustworthiness of the UC application. The requirements are (Baraki *et al.*, 2014; Hoffmann, 2014):

Information about functions

Users want to understand how the UC application works. To predict and anticipate the application's behavior, the user needs information about its functions. Hence, the application should inform the users about how and why they function in a certain situation. Although the results of using the application may be the same, the users tend to question a system's result if they do not understand how it has arisen. The functionality of the application should be easy to understand for the user. Applications should at least provide explanation for the 'how' question. Furthermore, the application should provide explanation for the necessity of required data input.

Explanation of Processes
To foster understandability as a determinant of trust, the application should inform the user about how it proceeds to reach a certain goal. Compared to the *Information about functions* requirement, the explanation of processes and algorithms goes more into detail and considers the technical realization. The level of explanation should be adapted to the experience and expectations of the individual user. However, in case of innovative algorithms it is necessary to find a trade-off between providing sufficient explanation and keeping corporate secrets.

Signaling the Function Status
This requirement aims at addressing a user's need for transparency. The application should provide an overview on which functions are executed as well as why and how they are executed. This should lead the application's behavior to be transparent for the user who thus better understands how the application works.

Level of Automation of Functions
One dilemma in designing trustworthy UC systems is to give the user enough control over the application while on the other hand establish a sufficient workflow. However, one major challenge in the development phase is to determine to which degree the application may act autonomously (without explicitly requesting user input) so that the users will not lose the feeling of control. Thus, to capture this dilemma the application should provide users control over the level of automation so that they can adapt it to their preferences and level of trust in the application.

Control of Processes
To complement the user's demand for control, the application should provide opportunities to intervene and control depending on its level of automation. This may include informing users about autonomously executed tasks and offering them an 'undo' functionality. In case that the user executes the task manually, it may be necessary that the system requires a confirmation that the user is aware of the consequences his actions may have.

Agreement to Functionality
To foster the user's perception of control over the UC application's functionality, it should request the user to confirm the functionalities. This request should appear right before the application was started for the first time. Furthermore, the users should be able to revoke their confirmation at any time.

Configurability
In order to build trust, the perceived adaptability of the UC application to a user's wishes and demands is an important criterion. The wish for a personalized system thereby inherits the need for a more competent system from a user's perspective. Thus, the functions of a UC application should provide options for personalization, since users may go different ways to reach their goals and the system should adapt to this circumstance by facilitating users to do so. The options for personalization should meet the users' demands. In addition, the configuration of the application should be intuitive to the user.

Assessment of Output

A user's perception of the output quality effects an application's trustworthiness. Therefore, users need an overview of how accurate and complete the information output is. In order to enable users to evaluate the output, the application should provide information about its accuracy and completeness.

Logging Processes
To increase trust by addressing transparency and non-repudiation, actions should be protocolled. It is thereby especially important for a user to know which process steps have been conducted autonomously without requiring user input. Hence, an application should provide an overview of both manually and autonomously conducted process steps.

For describing recurring requirements in a pattern structure, we followed a theory-driven approach. The requirements mentioned above have been derived from elements to build trust (also often called antecedents, dimensions, determinants or principles of trust) in information systems (IS), which can be found in respective IS literature. In the context of this work, we rely on work about trust summarized and enhanced by Söllner *et al.* (2012). However, since these trust-building elements are recurring and build the basis for the requirements, applying a pattern logic is applicable at this point. Figure 1 exemplarily shows how the requirement patterns have been documented. The structure is based on recommendations made by Franch *et al.* (2010) and complemented by Hoffmann (2014). The pattern structure has been developed as part of a dissertation project. It has been derived from relevant literature and was adapted to the context of trust, usability and jurisprudence. Due to space limitation, we cannot elaborate on the entire development process in this contribution. However, Hoffmann (2014) explicitly describes the approach of pattern (structure) development.

V-S-15: Control of Processes			
Meta Data	Goal	Users are able to control the processes of the application.	
	Trust Antecedent	*User Control* (Janson et al. 2013)	
	Dependencies	-	
	Relations	V-S-14: Level of Automation of Functions	
	Conflicts	-	
Template Success Notification	Standardized Requirement	The application should confirm the successful execution of processes to the user.	
	Extension	The application should confirm the successful execution of <process> to the user.	
		Parameter	Value
		<process>	[processes]
	Comments	Epecially for Sheridan-Verplank Level of automation (SVL) from 1 to 7.	
Template Withdrawal	Standardized Requirement	The application should enable the user to withdraw processes.	
	Extension	The application should enable the user to withdraw <process>.	
		Parameter	Value
		<process>	[processes]
	Comments	Useful for all SVL but SVL 10.	
Template Confirmation	Standardized Requirement	The application should confirm data input to the user.	
	Extension	The application should confirm <data input> to the user.	
		Parameter	Value
		<data input>	[data inputs]
	Comments	Epecially for SVL 1 to 5, where the users manually provide data input.	

Figure 1. Example of a requirement pattern - V-S-15: Control of Processes (Hoffmann, 2014)

5. SELECTING DESIGN PATTERNS

After eliciting the requirements for UC applications and transferring them to a pattern format, we needed to find design approaches that are most promising to meet them. We therefore followed a six-step process in which we involved various experts for requirements engineering, human-computer-interaction and design as well as domain experts. In the following, we present the steps of this process by exemplarily using the requirement introduced in section 4 (Control of Processes).

5.1 Ideate design alternatives

To find design principles that fit to the demonstrators' requirements, we conducted a workshop with experts in human computer interaction (HCI) to brainstorm suitable design ideas. The workshop was guided by the question, which design guidelines for the application could be identified to best meet the requirements. At this point, we set no limitations regarding realizability, potential effort or conflicts with other requirements or design alternatives. As a result, the experts described several design alternatives for each requirement. For the example requirement *Control of Processes*, three design alternatives could have been identified:

Control of Autonomous Adaptation
When the UC application autonomously adapts itself to a certain situation in order to provide new or better functionalities, users often experience a loss of control over the system's processes. Thus, users should receive a notification and be able to confirm or decline the context-dependent change of functions. In every case, it should be possible to withdraw the confirmation/declination at any time.

Emergency Button
One way to foster users' control of processes in which personal data is used to deliver new or better functionalities can be to provide the opportunity to stop personal data usage whenever the user wants. Therefore, an 'emergency button' should be implemented to intervene the collection and utilization of (peripheral) personal data. The user should be aware of this button and its functionality. It should be reached without spending much effort, either by opening the menu, the settings display or via shortcut. Pushing the button should open a pop-up window where the user is informed about the consequences. After confirmation, the UC application should run without collecting personal data anymore.

Level of Action Confirmation
To give users control over the applications' processes, users should be able to set a level of which actions they are willing to confirm and which ones they will not. Therefore, the settings menu should provide the option to configure the confirmation frequency depending on how critical an action is. The application should in turn adapt to this configuration.

5.2 Rate design alternatives

In a next step, the design alternatives had to be discussed, rated and ordered. We therefore reflected the overarching goals we wanted our application design to follow. From an ergonomic point of view, we aimed at developing *socially acceptable* applications as we understand the technical systems as parts of more complex sociotechnical systems, in which the user and the process of usage has to be regarded within the entire development process. Hence, we talked to HCI experts to assess the design alternatives regarding perceived usefulness and usability. In this step, we also addressed the *technical realizability* of the design alternatives. This includes both technical feasibility and economic (i.e., on a project management level) effort. To assess these values, software engineers and software project managers were consulted to give an appraisal. Furthermore, we needed to consider the degree of *requirement fulfillment*. Since the requirements were elicited literature-based, researchers and practitioners who have knowledge in trust-oriented software design have been invited to evaluate the design alternatives by challenging them against our requirements.

As a result, the idea *Control of Autonomous Adaptation* has been rated very useful by all stakeholders, since it provides an easy to implement and user-friendly alternative to provide process control to the user. An *Emergency Button* has been rated as a more complex (and thus more expensive) alternative, since all personal (and peripheral) data have to be identified in advance and intervention functionalities have to be implemented in many processes. However, from an HCI perspective this functionality has been valued as useful, since it provides an intuitive solution to the challenge of regarding a user's informational self-determination. The design idea *Level of Action Confirmation* has been valued as very complex to realize. The development process would need to include finding measures for criticality, defining all actions' critical level and design process modules that are active or inactive depending on the configuration. From a user's viewpoint, the result of implementing this design idea may be twofold: on the one side, it would improve the controllability of the process if configured right. On the other side, if the level of action confirmation would have been set to strict (e.g., by accident or not knowing what it does), the amount of confirmation pop-us may be counterintuitive and annoying to the user.

5.3 Select design alternatives(s)

After different experts had rated the design ideas, the goal of this step was to find suitable design alternatives for implementation by identifying potential conflicts, dependencies or relations among the design alternatives and other specifics outside of the respective requirement-design relation (i.e., design alternatives for other requirements). Therefore, requirement engineers and solution designers were confronted with the rating results.

As a result of discussing the examples mentioned above, *Control of Autonomous Adaptation* and *Emergency Button* appear to complement each other to address the requirement appropriately. They represent comparably easy-to-implement design alternatives with no conflicts within or outside the requirement-design relation. As opposed to this, *Level of Action Confirmation* is a more complex solution alternative for which the experts claim the effort will not justify the expected user need. Furthermore, this design alternative shows conflicts with actions autonomously conducted by the application, since a high level of action confirmation would possibly interrupt the applications' workflow repeatedly and make it unusable. Hence, *Control of Autonomous Adaptation* and *Emergency Button* were chosen as design guidelines, whereas *Level of Action Confirmation* was dismissed.

5.4 Implement design guideline(s)

In this process step, the selected design guidelines were implemented in our demonstrators Meet-U, Connect-U and Support-U. We thereby followed the development process described in the VENUS method. It basically contains the following steps (Hoffmann and Niemczyk, 2014):

- Describe *Use Cases* to comprehend users' needs for a certain functionality and their approaches to use the application
- Design *Data and Function Elements* visible to (and if needed editable for) the user (e.g., 'User profile' as data element and 'Edit profile name', 'Edit interests' etc. as related functions)
- Design *Workflows* for the use cases and *Sitemaps* to structure workflows
- Design *Function Layout* (i.e., wireframes) to transfer textual information and function elements to graphical elements
- Design *visual user interface*
- Develop *Prototypes* to enable evaluation

5.5 Evaluate design

The evaluation of the implementation aimed at assessing, whether the requirements have correctly and sufficiently been addressed by the design chosen. Therefore, prototypes have been developed and iteratively evaluated by experts to make adaptions as early as possible. The evaluation is essential to cast design guidelines into design patterns: only if a design is proven to sufficiently and correctly meet given requirements, we transfer it into a design pattern. In the VENUS method, evaluation and

validation methods include expert validation, simulation-driven evaluation and laboratory evaluation. A detailed description of the methods and their applicability for the development of UC applications is given by Hoffmann and Niemczyk (2014). For our applications, the design was evaluated and adapted until experts valued it appropriate to fulfill the given requirements.

5.6 Formulate design pattern(s)

To simplify and accelerate the interdisciplinary development process based on recurring requirements, enhancing design guidelines (i.e., solution) by information about the intent, affecting forces, the context and possible consequences will be helpful for context-dependent development. This enhancement results in reusable interdisciplinary design patterns. Whereas requirement patterns support the identification and documentation of requirements, design patterns help implementing a technical system design with regard to requirement fulfilment. The structure of the interdisciplinary design patterns is described in detail by Baraki *et al.* (2014). Table 1 provides an excerpt of the design patterns we used as an example for describing our process.

Table 1. Example Design Patterns (excerpt) (Baraki *et al.*, 2014)

Pattern name	Problem	Forces and Context	Solution	Consequences
Control of Auto-nomous Adaptation	Autonomous adaptations can result in usability problems. The goal of the pattern is to prevent the feeling of loss of control. Users may sense a loss of control if the behavior of an application is not comprehensible or if the behavior disturbs the current interaction with the application. The pattern helps to create understandable autonomous adaption and prevents the feeling of loss of control.	*Informational self-determination:* To support the user's self-determination in case of autonomous adaptation, the ultimate decision-making authority has to remain with the user - otherwise the system can adapt to unintended and irreversible states. *Transparency*: The autonomous adaptation is a black-box concept to the user. If the user does not receive information about next adaptation steps nor the possibility to govern automatically executed actions, he will experience loss of control and a missing overview on the different states and steps.	The user should be enabled to keep control of autonomous adaptations. This prevents the feeling of loss of control. Two cases have to be distinguished: 1) The user is currently interacting with the application. In this case, the application should notify the user about the upcoming adaptation and enable the user to determine if the application should adapt. The user should have a choice to accept, decline or delay the adaptation. 2) The user is currently not interacting with the application. This means that the adaptation can be performed. However, the application needs to provide the user an option to revert the adaptation. Adaptations with substantial effects on the system should be recorded in a history. Such a change may be the switching off of a surveillance system or of a ringtone. The adaptation design needs to be tailored to the application domain, development platform, and target user group. The cooperation with a usability engineer and/or a trust engineer is recommended.	The pattern is influenced by and influences the user interface design of the application. The adaptation notifications need to be integrated into the user interface design.

Emergency Button	This pattern should be applied if the application collects and uses personal data. It enables the user to halt collection and use of his personal data in a simple manner at any time.	*Informational self-determination:* The appliance of this pattern supports the user's right to informational self-determination by disabling any use or gathering of personal data by the application. It enables the user to maintain control of his/her own data. *Trust:* By providing a mechanism to the user to disable the collection and use of personal data, the user's acceptance and trust into the application can increase. This holds especially true in that situations where the user wants to be invisible to the application.	The implementation and the user interface design of an emergency button depend on the application domain and development platform. The button should be easily accessible at all times. It is important to give feedback to the user after activating the button. After pressing the button, the system stops immediately collecting and using personal data. Herein, all data from which other personal data can be inferred is included. If pressing the button impairs application functionalities, the application highlights these functions to provide visual feedback.	When pressing the button, all functionalities, which require personal data, need to be deactivated to prevent errors at runtime. The Emergency Button Pattern can be combined with the Enable/Disable Functions Pattern which addresses similar concerns.

As a result of our process, validated design patterns could have been identified that match the requirements (patterns) elicited from literature (Table 2).

Table 2. Requirement Patterns and related Design Patterns (Baraki et al., 2014)

REQUIREMENT PATTERN	DESIGN PATTERN
Information About Functions	On Demand Explanation
Explanation of Processes	Abridged Terms and Conditions
Signaling the Function Status	Trust and Transparency Control of Autonomous Adaptation
Level of Automation of Functions	Control of Autonomous Adaptation
Control of Processes	Control of Autonomous Adaptation Emergency Button
Agreement to Functionality	Emergency Button Enable/Disable Functions
Configurability	Enable/Disable Functions
Assessment of the Output	Context State Indication
Logging Processes	Data Access Log

6. CONCLUSION

This paper describes an approach of finding suitable design patterns based on requirements for developing socially acceptable UC applications as part of the VENUS method. Since the roles and dependencies of requirement and design patterns have not yet been clarified sufficiently in prior work, we aim at making a contribution to gain a better understanding to this. As a result of our work, we experience the consecutive utilization of design and requirement patterns as very useful to solve foundational design problems (e.g., to design trust-building elements). Admittedly, following a process similar to the one we described requires high operational and organizational effort, since many stakeholders have to be involved to define and validate the patterns. This circumstance, however, is counterbalanced by the effort saved when applying the patterns with minor adaptions regarding a specific problem context. We thus value an integrative approach in which design patterns are derived from requirements (patterns) as useful to bridge gaps in the early phase of system development, where recurring requirements call for similar solutions. Furthermore, we hope to assist practitioners in the

field of requirements and/or systems engineering by providing an example process that can be seen as a reference for developing, selecting and deploying design alternatives based on requirements.

7. ACKNOWLEDGEMENTS

We want to thank our Shepherd Andreas Fiesser for the valuable suggestions which helped improve our paper in the shepherding process. We would further like to thank the participants of the EuroPLoP'16 workshop: Padmalata Nistala, Nazila Gol Mohammadi, Johannes Iber, Jose Carlos Ciria Cosculluela, Ruslan Batdalov and Christopher Preschern. We benefited a lot from their comments and suggestions.

REFERENCES

Atzmüller, M., Macek, B.E., Hoffmann, A., Kibanov, M., Scholz, C., Söllner, M. and Stumme, G. (n.d.), "Connect-U – Development of Ubiquitous Systems for Enhancing Social Networking", in David, K., Geihs, K., Leimeister, J.M., Roßnagel, A., Schmidt, L., Stumme, G. and Wacker, A. (Eds.), *Interdisciplinary Design of Socio-technical Ubiquitous Systems*, Springer (forthcoming).

Baraki, H., Geihs, K., Hoffmann, A., Voigtmann, C., Kniewel, R., Macek, B.-E. and Zirfas, J. (2014), *Towards Interdisciplinary Design Patterns for Ubiquitous Computing Applications, ITeG Technical Reports*, Vol. 2, Kassel University Press, Kassel.

Baraki, H., Geihs, K., Voigtmann, C., Hoffmann, A., Kniewel, R., Macek, B.-E. and Zirfas, J. (2015), "Interdisciplinary design patterns for socially aware computing", *Proceedings of the 37th International Conference on Software Engineering*, Vol. 2, pp. 477–486.

Chung, E.S., Hong, J.I., Lin, J., Prabaker, M.K., Landay, J.A. and Liu, A.L. (2004), "Development and evaluation of emerging design patterns for ubiquitous computing", *DIS '04 Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*, pp. 233–242.

David, K., Geihs, K., Leimeister, J.M., Roßnagel, A., Schmidt, L., Stumme, G. and Wacker, A. (Eds.) (2014), *Socio-technical Design of Ubiquitous Computing Systems*, Springer International Publishing, Cham.

Franch, X., Palomares, C., Quer, C., Renault, S. and Lazzer, F. (2010), "A Metamodel for Software Requirement Patterns", in Wieringa, R. and Persson, A. (Eds.), *Requirements Engineering: Foundation for Software Quality*, *Lecture Notes in Computer Science*, Vol. 6182, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 85–90.

Geihs, K., Niemczyk, S., Roßnagel, A. and Witsch, A. (2014), "On the socially aware development of self-adaptive ubiquitous computing applications", *it - Information Technology*, Vol. 56 No. 1, pp. 1–41.

Göschka, K.M. and Haridi, S. (Eds.) (2012), *Distributed Applications and Interoperable Systems, Lecture Notes in Computer Science*, Springer Berlin Heidelberg.

Hafiz, M. (2013), "A pattern language for developing privacy enhancing technologies", *Software: Practice and Experience*, Vol. 43 No. 7, pp. 769–787.

Hoberg, S., Schmidt, L., Hoffmann, A., Söllner, M., Leimeister, J.M., Voigtmann, C., David, K., Zirfas, J. and Roßnagel, A. (2012), "Socially acceptable design of a ubiquitous system for monitoring elderly family members", *Proceedings of Sozio-technisches Systemdesign im Zeitalter des Ubiquitous Computing (SUBICO)*, pp. 349–364.

Hoffmann, A. (2014), *Anforderungsmuster zur Spezifikation soziotechnischer Systeme: Standardisierte Anforderungen der Vertrauenwürdigkeit und Rechtsverträglichkeit*, Kassel University Press, Kassel, Germany.

Hoffmann, A. and Niemczyk, S. (2014), *Die VENUS-Entwicklungsmethode. Eine interdisziplinäre Methode für soziotechnische Softwaregestaltung, ITeG Technical Reports*, Vol. 1, Kassel University Press, Kassel.

Lahlou, S. and Jegou, F. (2004), *European disappearing computer privacy design guidelines, Version 1.1*.

Landay, J.A. and Borriello, G. (2003), "Design patterns for ubiquitous computing", *IEEE Computer*, Vol. 36 No. 8, pp. 93–95.

Langheinrich, M. (2001), "Privacy by design - principles of privacy-aware ubiquitous systems", in Abowd, G.D., Brumitt, B. and Shafer, S. (Eds.), *Ubicomp 2001: Ubiquitous computing: International Conference proceedings*, Springer, Berlin, New York, pp. 273–291.

OECD, Organisation for Economic Co-operation and Development (2003), *Privacy Online: OECD Guidance on Policy and Practice*, OECD Publishing.

Robertson, S. and Robertson, J. (2006), *Mastering the requirements process*, Addison-Wesley Professional, Boston.

Ruiz-López, T., Noguera, M., Fórtiz, M. and Garrido, J.L. (2013), "Requirements Systematization through Pattern Application in Ubiquitous Systems", in *Ambient Intelligence-Software and Applications*, Springer, pp. 17–24.

Söllner, M., Hoffmann, A., Hoffmann, H., Wacker, A. and Leimeister, J.M. (2012), "Understanding the Formation of Trust in IT Artifacts", *ICIS 2012 Proceedings*.

Toro, A.D., Jiménez, B.B., Cortés, A.R. and Bonilla, M.T. (1999), "A Requirements Elicitation Approach Based in Templates and Patterns", *Workshop em Engenharia de Requisitos 1999*, pp. 17–29.

Wahono, C. (2002), "On the Requirements Pattern of Software Engineering", *Proceedings of the Temu Ilmiah XI*, pp. 1–7.

Weiser, M. (1991), "The computer for the 21st century", *Scientific American*, Vol. 265 No. 3, pp. 66–75.

Wieringa, R. and Persson, A. (Eds.) (2010), *Requirements Engineering: Foundation for Software Quality, Lecture Notes in Computer Science*, Springer Berlin Heidelberg, Berlin, Heidelberg.