Requirements Engineering for Context-Aware applications

Guido van der Zanden
g.vanderzanden@student.utwente.nl

ABSTRACT
Many context-aware applications are not widely spread due to a discrepancy between user expectation and the actual system behavior. Before this discrepancy is solved, context-awareness cannot fulfill the promise of providing calm technology that aids us with our daily tasks. In solving this discrepancy requirements engineering traditionally plays a role. We investigated the distinguishing properties of context-aware application in order to see how requirements engineering should be changed to fit context-awareness. As a result, we found user involvement is important, but low-fidelity prototyping is not adequate to do this. Software prototypes are better suited, but are costly to use in early stages of development. Furthermore, the environment needs to be analysed thoroughly, and quality requirements have to be specified in order to ensure functionality. Last, models are needed to describe the dynamic between context and functionality, which are already proposed in other research.

Keywords
Requirements engineering, context-aware systems, context-aware system development

1. INTRODUCTION
Since first introduced by Schilit and Theimer [1], the term ‘context-aware computing’ received a lot of attention. For example, context-awareness is one of the five key elements in the Ambient Intelligence vision [2], which foresees ‘calm technology’ that is unobtrusive and embedded into the background instead of ‘grey boxes’ it is nowadays. In order to develop calm technology, the technology needs to be aware of our context: where we are, who we are with, what our mood is etc. By being aware of the context, a system can be pro- or reactive towards users by providing information or services the user needs in a particular context. Instead of a tool, technologies evolve into pragmatic systems which support our daily lives.

Although many research is done in the field of context-awareness, context-aware applications are not widely spread. According to Sitou and Spanfelner [3], context-aware applications “seem to be particularly prone to problems related to a discrepancy between user expectation and systems behavior”. In other words, context-aware applications do not do what users expect them to do, therefore instead of being a helpful application becoming an application ignored by users.

In the development of applications, this discrepancy is dealt with by requirements engineering. In short, requirements describe what a system or products must do [4]. Furthermore, requirements are a manner of communication with developers and clients [5]: the developer knows what he must implement, and the client knows what he will get. Throughout the development process the system is evaluated to see if it meets the requirements. In user-centric design, requirements are less static. In an iterative design process, users are constantly involved. By involving users in the design, they are more able to express what their needs from the system are, thus improving the requirements and application [6]. In Section 3 we will shortly discuss requirements engineering in more detail.

In the development of context-aware applications, proper requirements engineering is as important as it is for traditional applications. Developers face challenges because context-aware systems are different from traditional systems, having “different features such as heterogeneity, high complexity, and artificial intelligence” [7]. To handle those challenges, requirements engineering methods are needed which address those challenges. Therefore, we argue that by identifying the distinguishing properties of context-aware applications we can identify what is needed to do proper requirements engineering. The distinguishing properties of context-aware applications are discussed in Section 4. In Section 5 we will elaborate on the influence of the properties found on requirements engineering.

2. RELATED WORK
Choi [7], Desmet et al. [8] both put effort in developing requirements engineering methods, both achieving a similar result. Based on their interpretation of context, Choi [7] developed a context-aware use case diagram, a context-switch diagram and a dynamic service model. All three have the goal to let stakeholders understand the working of the system and the context on which it is based. The context-aware use case diagram in short extends normal use case diagrams with context-aware behavior in order to express the influence of context on a system easily at a high level. The context-switch diagram combines the UML class and state diagram in order to express transitions between context. In example, the diagram expresses the transition from normal to busy with an arrow which holds the condition ‘job’. Vice versa, the condition ‘not job’ expresses the transition from busy to normal. The dy-
namic service model is developed in order to let stakeholders understand the internal processes of a context-aware application.

Desmet et al. [8] developed a comparable model, but specify context-aware behavior in only one diagram, the context-oriented domain analysis diagram. The diagram specifies one action, i.e. an incoming call in a tree-like diagram. In the diagram, behavior is divided into basic and context-aware behavior and basic behavior is always on top in the diagram. The diagram must be read top-down and for every step down in the context-aware behavior part, multiple options are available. The diagram provides some vocabulary to express context-dependent adaptations.

Whereas both researches mentioned above provide tools to write down context-aware behavior in a simple manner, Sitou and Spanfelner [3] consider requirements engineering in a broader way. Firstly, they describe a model of usage context, consisting of the user, task, domain, platform, dialog and presentation model. Those models serve as template to describe possible context at a high level. In the process of finding the actual requirements, they propose to split up the process. In the first stage, all needed functionality has to be discovered and divided into context dependent and context independent functionality. In the second stage, the context dependent behavior is divided into behavior that works 'out-of-the-box' and behavior which needs user calibration. Stage 1 and 2 are gone through iteratively and in stage 2 methods like contextual analysis are used to further analyze the user's behavior.

Kokos-Mazuryk et al. [9] investigates the development of requirements engineering methods for pervasive services. Although not the same, context-awareness is an important factor for pervasive services. Their approach consists of describing the distinguishing factors of pervasive services, which are divided into contextual and non-contextual properties. However, their research is still in an early phase. Therefore only their envisioned research approach is known.

3. REQUIREMENTS ENGINEERING
A requirements specification is produced early in system development. This document describes what a system must do. After an analysis phase in which stakeholders and developers figure out what their needs from the system are, the requirements specification is written. The specification is checked by the stakeholders after which the specification serves as input for the design phase, in which the system is programmed. At the end of development, the system is tested to see if it fulfills the requirements [10]. In short, actions that have to be undertaken regarding requirements are discovering them (elicitation), writing them down (formulation) and checking if they are correct (validation).

There are different kinds of requirements, Lauesen [10] divides requirements in three categories:

- **Data** requirements specify what data is stored in databases and the format of input and output.
- **Functional** requirements specify the functions of the system, how it records, computes, transforms and transmits data.
- **Quality** requirements specify quality factors such as performance, usability or maintenance.

The process described above, the waterfall model, is an ideal. When it would be applied strictly, which was practice many years ago, “requirements are gathered, analyzed, negotiated, and rigidly defined” [6]. Ferre and Medinilla [11] argue that in traditional software engineering emphasis was on descriptive complexity, the “quantity of information required to describe the system” [11]. In other words, emphasis was on developing a good systems from the software engineer’s point of view. User-centric design rejects this vision and in the human-centred design framework (ISO 13407), four key principles are found [12]:

- The active involvement of users and clear understanding of user and task requirements.
- An appropriate allocation of function between user and system.
- Iteration of design solutions.
- Multi-disciplinary design teams.

Based on the four key principles, the ISO standard prescribes five essential processes in user-centric design, which are shown in Figure 1. To ensure active involvement of users, cycle iterations have to be short. For instance, De-Bellis and Haapala [6] aim to “deliver enhanced versions of the prototype every eight week, with changes based on user feedback”. In user-centric design requirements are enhanced after every cycle, thus requirements are constantly evolving. In methods proposed by user-centric design, no methods are found to write requirements down. The methods, such as focus groups or scenarios, help developers to elicit requirements.

In requirements engineering elicitation and formulating requirement are two separate task, therefore both need attention in developing requirements engineering for context-aware applications. Because the methods to formulate requirements are a ‘tool’ for communication, both the user and developer need to understand them in order to communicate about the system. Simplicity of the models is therefore important, but also be expressive enough for developers in order to be the basis on which the system is implemented.

4. DISTINGUISHING PROPERTIES OF CONTEXT-AWARENESS
In finding distinguishing properties of context-aware applications, we investigated several definitions of context-awareness and context. Because in context-awareness output is based on context, both input and output differ significantly from normal applications.
4.1 Context-based Output

The most obvious distinguishing property of context-aware applications is the ability to provide output based on context information. According to Dey et al. [14]:

*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevance depends on the user's task.*

The definition holds three main elements, namely context, providing information and/or services and the user’s task. Context will be discussed in the following subsection, because context is the input for context-aware application. Here we will focus on the later two elements.

Providing information or services is a normal job for a system, and are triggered by explicit user input. Context-awareness will change the usage scenarios of applications because context takes over the input part. According to Dey et al. [14], there are three ways in which an application can respond to context:

- **Presentation** of information and services to a user
- **Automatic execution** of a service
- **Tagging** of context to information for later retrieval

In example, presentation of information could be providing the user with location specific weather information. Execution of a service could be automatically printing on the nearest printer, whereas presentation of services could be displaying the nearest printers, thus leaving the choice up to the user. Tagging is “the ability to associate digital data with the user’s context” [14]. The data can be viewed when an user is in that associated context. In example, an user could leave a note at the fridge saying it is out of milk. When he or someone else uses the fridge, he will see the virtual note.

Dey et al. [14] stated relevancy to be dependant on the user’s task. However, Bradley and Dunlop [15] argue that context-aware application can also help users with information or services not related to their tasks. The application uses context to “infer future user intentions of which the user may be currently unaware” [15]. For example, when walking home from work, the application may inform the user that a friend is in a nearby cafe.

4.2 Context input

Context-based behavior is dependant on context in order to operate, and it is unlikely the application itself will gather this information due to cost, complexity, efficiency etc. [16]. Instead, context-aware application will gather context from external context sources. Therefore, context-aware applications are dependant on the environment they are used in. The environment is dynamic, simply because it is unknown where the application will be used, or because it is a mobile application. Therefore, the input of context-aware applications is also dynamic: there is no guarantee that context information is available or that the information is sufficient to base an adaptation upon.

If no context information is available, no context adaptations can be made. However, if context information is available, there is no guarantee it is usable. For example, the user location measured by GPS is far more accurate than by cell-id, and the output of the application is affected by this. This fluctuations in context information can be measured by Quality of Context (QoC), a concept described by, among others, Sheikh et al. [17]. QoC measures the quality of context information by several indicators, i.e. precision or freshness. In general, if the QoC is low and the adaptation level is high, the application is likely to adapt differently than the user expects. Vice versa, if the QoC is high and the adaptation level is low, the application missed opportunities for adaptations. Therefore, a context-aware application does not only bases its output on context, but also on the quality of the context.

The context available and the quality of that context determine if and how the application can perform an adaptation. However, the context available is typically low-level information, e.g. a sensor measures the velocity of an user rather than measuring he is driving a car or walking. Therefore, in contrast to normal application, context-aware applications deal with implicit input. Normal application react on explicit user input, e.g. double clicking a desktop icon, whereas context-aware applications have to reason with the input. If the application is programmed to block incoming calls when the user is in a car, it cannot simply sense that it is in a car. Instead, it has to combine context information such as velocity or appointments of the user to reason he or she is in a car. Instead of input-output, a context-aware application does input-reason-output.

5. REQUIREMENTS ENGINEERING FOR CONTEXT-AWARENESS

In the preceding section we have shown context-awareness is different in three ways. First, output is context-based instead of a reaction to user input. The input itself is also different, being implicit and not always available. Together they form a chain from gathering context, reasoning with context towards presenting information or executing
services. In the following subsections we propose three factors which are important to address those context-aware specific issues.

5.1 User involvement

Context-aware application present information or execute services based on context information. Therefore, when specifying what the application must do, one must also specify when the application must do this. Also, according to Dey et al. [14] a difference can be made between presenting information, presenting services and executing services. Next to what and when, sometimes users must specify how they want their application to work.

With automatic execution or presentation it is “likely that usability problems occur” [3]. Context-based output when not expected by the user (e.g., presenting nearby friends when at work) is a typical example, caused by wrongly interpreted context information or, when context information is correctly interpreted, an incorrect coupling of a contextual situation and output. Knowing exactly when an user expects what from a system is therefore crucial to avoid usability problems. Knowing how is less important, because only a few possibilities are possible (presentation, execution or tagging).

In requirements engineering for normal applications, users have difficulties expressing what they need, specify solutions instead of demands etc. [10]. For context-aware applications, users also have to specify when and how, and there is no reason to assume this is less difficult for users. In user-centric design, users are involved in the development of the application to overcome this problem. According to Pekkola et al. [18], “users cannot always predict nor articulate their own needs for a system” and emphasize that prototyping needs to be done with high-fidelity prototypes, as part of the user’s daily activities and for a longer time in order to help users fully understand the system and their needs from the system.

In the requirements phase of user-centric design, methods like scenarios are used to help users understand the system. Scenarios are “detailed realistic examples of how users may carry out their tasks in a specified context with the future system” [12]. Primary aim is to help users understand the system. Storyboarding, a method which uses pictures to represent a scenario, is already used on the Mime Project website to demonstrate a couple of context-aware prototypes [19]. However, based on a story or some pictures, it is still hard for users to know if they want the functionality described. For example, blocking incoming calls when in a meeting seems reasonable. However, when actually missing a call from your pregnant wife, judgment will be different. No specific methods are available in the requirements phase of user-centric design to create an as real as possible experience for the user.

In user-centric design, validation of requirements is also part of the design phase. In this phase the system prototypes are made and used for testing. Because user-centric design is an iterative process, in early stages of development low-fidelity prototypes will be used. Those prototypes typically are screenshots of the application. In paper prototyping those screenshots are presented to the user, who will tell his next action. A developer presents a following screenshot according to the action of the user. The user will give feedback which will be used in the next development step. However, context-aware functionality cannot be captured in screenshots, because they cannot express a contextual situation. This could be solved by letting a developer explain some contextual situation and afterwards showing a screenshot of the application. Nevertheless, it still remains questionable whether the user can fully understand the system in this manner.

After specifying requirements, the best way to validate and improve them is by using high-fidelity prototypes in real life situations, as stated above. For example, a mobile shopping assistant should be tested by users with a real application who really intend to buy something. High-fidelity prototypes are in general implemented applications, and require “clearly defined total structure, distinct functional boundaries, and an agreed interface style” [12], therefore typically used in late stadium of development. If used in early stage of development, many cost are made in implementing the application and if the structure, boundaries or interface changes, many cost are made in reprogramming the application. High-fidelity prototypes are the best manner to discover and improve requirements, but cost may be disproportional in early phase of development to discover requirements.

5.2 Models

Models in requirements engineering are used to write down user requirements. These models are used to check with users whether the requirements are captured properly, and by programmers to implement the application. For example, a data model can easily be checked by an user to see if all data is captured. The programmer uses this information to design the database and to check whether the design is complete.

For context-aware applications, modeling context is needed. Such a model need to express a contextual situation and the linked output. Choi [7] and Desmet et al. [8] both proposed a context diagram for usage in requirement engineering, both proposing a similar model. In their diagrams, high-level context determines output of the system. For example, an incoming call is treated differently when the user is in his car, at home or at work. In the diagrams, the function ‘incoming call’ would be the root node, the location based adaptations are three separate child nodes. This approach seems obvious to use; it describes what the system must do without the use of complicated context models, therefore easy to understand for users. Furthermore, the models do not make assumptions about the implementation.

The context models show what applications do in different contextual situations. This autonomous behavior takes control away from the user, which is likely to cause usability problems (e.g., a service is executed automatically while the user would rather do this himself). Therefore, making explicit the division of responsibility (e.g., who will start the service) is important to discover usability issues, test and solve them.

For normal applications the allocation of function chart is
used, which is done in order to optimise the overall process by dividing tasks between users and the system as optimal as possible [12]. In the model, every vertical strip represents an user or the system. Tasks are divided between users and system, and information flows between them through an interface. The model can easily be extended to be used for context-aware application, as shown in Figure 2. The ‘actor’ context is added, and the interface between the application and user indicates whether the application task is executed manually by the user, automatically executed by the application or only information is presented automatically.

In the figure, a shopping assistant presents special offers to the user when he or she is browsing through the woman’s-wear department. Likewise, the model can express that special offers are only presented when the user asks for them, after which the application fetches special offers based on the context. One could furthermore model if the shopping assistant is executed automatically when the user enter a shop, or if he or she explicitly has to start the shopping assistant.

As last, above we stated that the context models describe high-level context. Obviously, low-level context would much harder for users to understand. Furthermore, because to gather high-level context information, low-level context has to be reasoned with. Reasoning with context is something to do in the design and implementation phase, because it determines how an application delivers its functionality. Therefore, only high-level context information should be used to specify functionality.

5.3 Environment Requirements

The environment in which an application runs determines if and to what extent the application can deliver context-based output. If no context information is available, no context adaptation can be made. If context is available, the application must determine how useful the information is. Depending on the quality of context, the application changes the adaptation level. The adaptation level a context-aware application delivers, is a indication of the quality delivered. Therefore, when specifying requirements about the environment, quality requirements are specified.

In requirements engineering for traditional applications, quality requirement specify non-functional requirements. For example usability, efficiency and maintainability can be specified in terms or clicks needed to perform a task or the maximum time the application may spend on a calculation [10]. To meet the quality requirements, effort has to be made in both soft- and hardware. If a new invoice system runs slow on the employees workstations, workstations may need to be upgraded.

Quality requirements need to be specified with care. When specifying to guarantee an application has a maximum response time of 1 second, costs will be very high to fulfil the requirement. The same goes for context-aware application: specifying the application will work everywhere at anytime can lead to insuperable technical and financial problems. A compromise must be made between functionality and feasibility.

When determining which quality factors are important for normal applications, a quality grid can be used. In the quality grid, quality factors (such as usability, efficiency etc.) are assessed with regards to their importance. Key is not to make everything important, but to pick out the really important ones, to be balanced [10]. In a similar way, context-aware functionality can be assessed with regards to different environments. For example, an application which notifies the user of nearby friends should work good in a city, can perform less well on the countryside and not work at all in a forest. The environment ‘city’ will be critical for the application, while the countryside is treated as usual and the forest is ignored.

To assess the application with regards to the environment, one must first know the environment. In development of normal application, before the requirements engineering phase an assessment is made of context of use. In the context-of-use analysis an overview is made of the user group, tasks, technical, physical and organisational environment [12]. For context-aware applications, mainly the technical environment is important. Characteristics like bandwidth and devices or sensors nearby are important [9]. Furthermore, physical characteristics are important because they can influence sensor and communication abilities. However, those characteristics can change and it is not possible to know every environment the application will be used in. To overcome this, ‘environment personas’ could be used. Normal personas are “caricatures to represent the most important user groups. Each persona is given a name, personality and picture” [12]. Personas are useful when there is no or little direct access to users. An environment persona would represent the environments (e.g., at home, at the office, at the mall) which are most important for the application. The relevancy of environments depends on application, a shopping assistant only
has purpose in a shop, therefore other environments can be ignored.

After assessing how important it is to have functionality in a certain environment, one can decide what functionality should be guaranteed in the different environments. In the friend finder example, in a city area friends are located at building level (e.g., the application indicates a friend is in Cafe X), while at the countryside friends are located at street level. Specifying this kind of quality requirements will oblige you to adjust the environment, how this is done is relevant in the implementation of the application.

Furthermore, specifying quality requirements means functionality has to be specified at different levels. The requirement ‘The application shall locate friends of the user’ is branched into locating friends at building or street level. This functionality should be treated as normal functionality, choosing those different levels will require user involvement. In writing down the different levels, the quality of context indicators are useful.

6. CONCLUSIONS AND FUTURE WORK
Context-aware application distinguish themselves from traditional application because of the input they use, how they process input and their output. Context-based output makes applications more autonomous in their behavior. By presenting information or executing services, the application can support the user. Furthermore, the input of a context-aware application is dynamic. While normal application receive unambiguous from the user, a context-aware application receives information from context sources. The context information has to be reasoned with to find meaningful high-level context information. Furthermore, the context can differ in amount and quality, thereby influencing the context-based output of the application. This is because context-aware applications rely on external context sources and there is no guarantee those are present in the environment.

To handle the differences in requirements engineering, we proposed three points of attention. First, discovering what users want from the system in which context needs user involvement, as described in user-centric design methodology. This is because context-based output is complex functionality and it is hard for users to predict their needs from the system. However, only high-fidelity prototyping is adequate to create an as real as possible experience for the user. The disadvantage of using high-fidelity prototypes in the early stages of development are high costs and high risks.

Secondly, models are needed to write down the discovered requirements. Models to describe context-aware functionality are already developed. Also, we propose to use an allocation of function diagram, in which explicitly is recorded whether the user or the application triggers a task. By doing this, different variants are discovered and can be tested in order to pick the most appropriate.

Third, in requirements engineering one must deal with the dynamic environment in which the application will be used. The environment determines whether context can be gathered, and what the quality of this context is. The environment is therefore directly related to the quality the application can deliver. Choices have to be made which quality has to be guaranteed in the different environments. Therefore, functionality has to be specified at different levels of context quality. Furthermore, choices made will oblige the developer to adjust the environment in order to guarantee the context delivery.

The differences found do not enforce dramatic changes in requirements engineering. Existing concepts such as user involvement and analysis of the context-of-use can be applied. Those methods and techniques can be used in addition to other, already existing methods. In comparison to normal requirements engineering, emphasis will move somewhat to technical requirements which describe the environment in which the application will be used, because the technical environment is of great importance to the functionality the application can deliver.

For future work, more thorough research should be done on the separate distinguishing properties of context-awareness. For example, methods to create high-fidelity prototypes with less costs and risk are very useful in requirements engineering for context-aware applications. We did not investigate in detail methodologies like Rapid Application Development (RAD) [12], which aim to develop such high-fidelity prototypes at low costs and risks. Interesting to investigate is if those methodologies also are suitable for context-aware application, being highly complex and have to coop with heterogeneity [7].

Next to user involvement in requirements engineer, we think the environment will have to play a big role in requirements engineering. The environment is more important for context-aware application than for normal application, therefore less attention is paid to the environment in the past. As the research on context-awareness continues, the results about architectures and infrastructures for context-awareness should be investigated and integrated in the requirements engineering process. By doing so, methods for requirements engineering can be developed which help requirements engineers to formulate requirements regarding the environment. If done properly, developer can more easily convert requirements into a system design and eventually into the implemented system.

References


