Incorporating medical knowledge in the design process of context-aware well-being systems

Steven Bosems and Marten van Sinderen
Faculty of EEMCS, University of Twente, The Netherlands
{s.bosems, m.j.vansinderen}@utwente.nl

Abstract—The concept of self-adapting applications delivering enriched end-user services is highly promising. These context-aware applications can be seen in increasing numbers in a wide range of fields, including that of health and well-being. Using sensors to collect context data and smart reasoning algorithms to deduce higher-level information, they have the potential to adapt their behavior in real time to better suit the context at hand. However, the development process for these types of systems is challenging, as the user needs and preferences in different contexts are not easy to anticipate at design time. If services offered by the application do not align with the user needs in some situations, the user may disregard the application altogether. In order to address this challenge, we propose to use medical knowledge related to well-being in the development process of context-aware well-being systems. We believe that this approach allows for improved identification and design of effective context-aware services for well-being support.

Index Terms — Context-Aware Applications, Well-Being, Domain Modeling, Model Driven Development

I. INTRODUCTION

Mobile phones sold today are equipped with an array of sensors capable of measuring the context around them. Software applications that can alter their behavior based on these sensor measurements have great potential to offer added-value services to end-users. Consequently, these context-aware applications can be found in a growing number of domains, including smart environments and well-being. Well-being is defined as the state of being comfortable, healthy, or happy.

The benefits of context-aware applications for well-being are evident: services provided to the user better suit the current situation than would be the case with applications that are not aware of their context. However, this class of systems is still new, developers and users alike having little to no experience to this adaptive behavior. As such, the development of the applications is challenging. Especially with context-aware applications provided on mobile devices such as smart phones, the context is continuously changing, which adds to the uncertainty of the behavior. To address these challenges, we propose to use a domain-specific model incorporating medical knowledge on how different variables in the well-being domain are interrelated to guide the development process of context-aware well-being applications.

In this paper, we will elaborate on the structure and contents of the domain model, discuss the process in which it is to be used and provide an example instantiation of this process. Furthermore, we propose the use of model transformations to facilitate automation in this process.

This paper is structured as follows: section II discusses the problem faced when developing context-aware well-being applications, section III elaborates our approach to dealing with this problem, section IV describes the possibility to use model transformation techniques, section V compares related work, and section VI gives concluding remarks and directions for future work.

II. PROBLEM STATEMENT

Current context-aware applications promise services that adapt to the user’s needs depending on the context the user is in. These promises, however, can not always be kept. The context of the user is continuously changing and in order to make a decision about the services to be delivered, numerous assumptions have to be made by designers while developing the application. When the application is put in practice, these assumptions do not always hold, resulting in application behavior that does not match the expectations or demands of the user. Users and designers alike can not foresee every possible context situation the application will have to operate in. Because of this, clear system goals and user requirements can not be formulated up front. With this type of application being innovative, users are unlikely to have experience with comparable products, adding to our design problem.

III. APPROACH

With traditional software development methods unsuited for context-aware well-being applications, we propose to use an iterative development process. This process is guided by causal domain models of the well-being domain. Previous approaches have primarily focused on the development of methods that are to improve the collection of user requirements; examples of such methods are mobile requirements engineering and “play testing” [1], [2].

A. Domain modeling

The domain of well-being is both structured and dynamic. It consists of elements with continuously changing properties due to interactions between them. Modeling languages that are common for software engineers are unsuitable to capture this: they either allow for the representation of the static structure of the domain, i.e. defining the classes, their attributes, operations, and relationships among elements, or they allow to capture the behavior of elements or the interaction between elements. A combination of these, as is needed when trying
to analyze causal relations in the well-being domain, is not provided.

To capture both the structure and the causal relations between domain elements, we use the causal loop diagram (CLD) language [3]. A CLD consists of variables representing context factors and well-being conditions, and arrows between them, indicating either positive or negative causal relations. A positive causal link between A and B indicates that for an increase of A, B will increase too. A negative causal link means the opposite: an increase in A will result in a decrease of B.

In the well-being domain, we can see a distinction between two types of elements: those that are physical and those that are conceptual. Physical elements reside in the real world: they can often be observed or measured directly using sensors. Conceptual elements are abstract ideas and have to be deduced from physical observations as they cannot be measured using sensor. For example, blood pressure is a physical element, whereas stress is conceptual. Using traditional CLDs, we cannot capture this distinction. Using annotations in the CLD, this distinction can be captured. We introduce three annotations:

- **Observable** [o] Variables with this annotation can be directly observed or measured using sensors.
- **Derivable** [d] These variables cannot be directly observed, but have to be derived from measurements.
- **Controllable** [c] Variables of this type can be directly controlled by the application using actuators.

The extended CLD language allows for causal reasoning over the elements of the well-being domain. Using causal reasoning, we can on the effects of changes in the domain. This means that we can deduce what aspects of well-being should be affected in order to improve the well-being of a person. Taking the annotations into account, we can reason what should be measured, and what actuators should be used to influence the user. Fig. 1 illustrates how this deduction process works: if we want to improve the feeling of well-being of a person, we should lower the amount of stress experienced. We see that stress can be decreased by increasing productivity or decreasing the perceived workload. Both of these can be achieved by decreasing the number of task switches. Following this reasoning process, we see that a decrease in the number of task switches causes an increase in the feeling of well-being.

To create a causal model for the well-being domain as a whole, we looked at medical literature and conducted expert interviews. Medical works such as [4] present us with the workings of the human body, providing us with norm values regarding its operation; these norm values are given as upper and lower bounds for vital signs measurements such as heartbeat rate and blood pressure. The International Classification of Functioning, Disability and Health (ICF) [5] gives an overview of the structure of the well-being domain, but no causal relations between elements are given. In order to obtain these, we interviewed experts in the fields of both mental and physical well-being. Part of the resulting model can be found in Fig. 1.

### B. Development process

The proposed domain model for well-being can be used to guide the development process of applications that aim at improving aspects of the well-being of people. We define a four-step process which moves from the general well-being domain model to a first prototype of the application. The prototype can be used to elicit specific application requirements from the user.

1) **Domain model reduction:** The overall causal domain model for well-being is the primary input for our development process. However, as this model covers the entire domain, it is infeasible to use it for the development of an application that focuses on specific well-being support and thus only has to consider a part of the well-being domain. As such, the development process starts by reducing the overall domain model to a domain model that is specific to the application under development. Firstly, the goal of the application has to be decided on. This goal will include an increase of the feeling of well-being, but should also include the means through which this is to be achieved. The application specific domain model (ASDM) should include observable variables; only by including these can the application be made aware of its context.

2) **Application structure specification:** Once the ASDM has been defined, it can be used to start the specification of the static application structure. An example modeling language that can be used for this purpose is the UML Class diagram notation. This model captures what components the application is made up of, how they are related, and which functionality they are to offer. As a basis for this model, a high level architecture such as defined in [6] can be used. During this step, it is to be decided whether the application is to be stand-alone, or whether it is to function as a part of an existing, shared infrastructure. In case of the former, interaction with the context, e.g. the communication with sensors, data storage and the use of actuators, is to be implemented in the application itself; in case of the latter, functionality regarding sensing and external communication may be offered by the infrastructure and in which case it does not have to be provided by the application.

3) **Application behavior specification:** Where the static application structure defines how components are related, the behavior specification describes how these components interact and what processing is to be performed by the different components. This behavior can be modeled using the Business Process Modeling (BPM) language, or UML Interaction overview diagrams may be used. In this paper, we will use the former.

The application behavior that is to be modeled is three-fold. To start, a description of the interaction between the application and the context is needed. Here, the application is
regarded as a black box, only focusing on the input obtained and the output produced based on this. When this has been described, a more detailed description of the system behavior can be made. Next, the communication between the different components is to be described. This behavior includes a discussion on which component initiates the communication and what the response would entail. The final step in discussing the behavior requires a detailed description of the internal workings of the components, capturing the computational steps performed.

4) Prototype development: The final process step to be taken is the combination of the models created in previous steps and developing a prototype application. Additional information is required regarding the way the application is to interact with the user: the models created so far only capture if and when interaction is to take place, but the best way of doing so is not yet described.

Once a prototype application has been created, it can be provided to a user to test it. During this trial, the user will elicit new requirements for the application. Using these, the process can be reiterated, adapting the models created during the various steps. This reiteration is to be performed until the prototype converges to an application that can be put to the market.

C. Example
To illustrate the development process discussed, we shall provide an example. In this example, we will describe the design process of an application that is to reduce the amount of stress in the user, resulting in an increased feeling of well-being. We will describe the steps of the process that will lead to a prototype.

1) Domain model reduction: In the application under development, we focus on stress reduction to improve the feeling of well-being. These two elements are added to our ASDM. From the overall domain model, depicted in Fig. 1, we can deduce that stress is caused by, among others, the perceived workload of the user and by the user’s productivity. These variables and relations are also added to the ASDM. As these two domain elements cannot be directly influenced, we need to extend our model; both productivity and the workload perception are influenced by the number of task switches a user performs while working. It is general knowledge that humans cannot truly multitask as this decreases productivity and increases the perceived workload. We therefore include this variable in our domain model. Now we can advise the user regarding his task switching behavior. However, what is not possible yet is to measure if an advice should be given. To deduce whether the application should provide feedback to the user, and to know if this feedback has been adhered to, we have to measure certain well-being properties. Two domain elements that are affected by the amount of stress are the heart rate and blood pressure. Both of these can be measured by sensors. What can also be measured, is the number of task switches a user performs while working; although this is measured by software, rather than hardware, we regard it as a sensor. With the ASDM containing our goal, ways of influencing the user, and means to measure the context, we consider it complete and move to the next development step. The model as described here is depicted in Fig. 2.

2) Application structure specification: The application structure captures the different components of the application and how they relate. We base the structure of our stress
management application on the high level architecture for context-aware well-being systems described by [6].

The first components we include in the static structure are the sensors. By looking at our ASDM, we see that three sensors are required: a heart rate sensor, a blood pressure sensor, and a sensor for the number of task switches. Although this last element is not measured by a physical sensor, there are applications available that can provide us with the required information, such as [7].

Next, we have to be able to interpret the raw sensor data that has been collected in order to reason about abstract concepts. Looking at our ASDM, we identify four abstract concepts: feeling of well-being, stress, productivity, and perceived workload. As the assumption is that a decrease in stress results in an increase in the feeling of well-being, we assume the latter is implicit in our application. We add the other three variables as classes to our model.

To interpret values for our abstract concepts, we need a component to combine information from the sensors. We add a DataInterpretation class to the model that is to perform this task. This class is then connected to two abstract classes Sensor and Concept. The former is extended by the classes HeartRateSensor and BloodPressureSensor, the latter is extended by Stress, Productivity and PerceivedWorkload. The data interpretation class also verifies whether the read and deduced information is within the normal bounds.

Now we have interpreted data, actions have to be undertaken if the data values are no longer considered normal. This is done by a PlanAction class that is related to the DataInterpretation class on the one hand, and to an abstract Actuator class on the other. As our ASDM does not include controllable elements, we will present actions through a user interface, which is modeled in the UserInterface class that extends the Actuator class. The resulting static application structure model can be found in Fig. 3.

3) Application behavior specification: The behavior of the application can be described on three levels: (i) the application as a whole, (ii) interactions among components, and (iii) the internal workings of components.

At the highest level, the stress management system we are developing is to use the number of task switches, the heart rate, and the blood pressure sensors to obtain information about its context. If all three are deemed too high, based on medical and personal norms, the system is to show a message on the user interface informing the user that a high number of task switches can cause stress and that it is advised to be more focused.

Next, we look at the interaction among components. To be able to perform, the system is to use data collected by sensors. This is done in a “pull” fashion: the DataInterpretation component queries the sensors with a given frequency. The alternative would be to have the sensors push new data as soon as it arrives, but this might be too calculation intensive for our application. When the data has been interpreted, it is sent to the Storage component, and to the PlanAction component. The latter performs its calculation and sends the result, i.e. what type of action is to be undertaken, to the UserInterface, which displays the message to the user.

The final step in the definition process for the application behavior entails the description of behavior per component. We will elaborate this step by describing the reasoning process for the DataInterpretation component. The process starts with the component requesting the current measured value from the three sensors. When these values have been received, the component has to reason about them. Based on medical literature such as [4], we know that a heart rate should be between 60 and 100 beats per minute for an average adult in a resting state. The blood pressure should be between 160/95 mm Hg and 90/60 mm Hg, the optimum value being 120/80 mm Hg. Looking at the number of context switches, we can identify a scale ranging from task under-load to task over-load. The read values should also be compared to the historic data collected: if our reading is within the normal range based on literature, but not normal when considering the user’s history, action has to be undertaken too.

If the number of task switches is considered high we can reason, using our ASDM, that the perceived workload is also high, causing stress. The heart rate and blood pressure can be used to confirm this. When the conclusion about the stress level has been drawn, the DataInterpretation component signals the PlanAction component that the user should be influenced to reduce stress. This conclusion is also stored using the Storage.

Once we finished this reasoning process per component, we can continue with the prototype implementation once this has been done. This will result in a diagram such as given in Fig. 4.

4) Prototype development: The final development step entails the programming and development of the prototype application. We will not discuss this process, as it consists of combining the design choices made in the previous steps and writing application code for it.

IV. MODEL TRANSFORMATIONS

As we use models as our primary development artifact, we see the possibility for automating the development process of context-aware well-being applications. We think this can be achieved by incorporating techniques from the model-driven
engineering (MDE) paradigm [8]. A technique proposed in MDE, is the use of model transformations. Using rules that are based on regularities found in the development process, i.e. the steps taken when moving from one model to the next, it is possible to partially automate the process. As design is a creative process, it will be unlikely to fully generate a prototype application from the causal domain model; manual input will still be required.

During the first development step, i.e. choosing the application goal, we immediately require human intervention: we can not automatically chose which type of system to design. However, it is possible to aid in the selection process of the related variables, making sure the selected set is complete.

Going from an ASDM to the static application structure can again be partially automated. We assume part of the architecture to be similar for all context-aware well-being applications: abstract classes Sensor, Concept and Actuator, the classes DataInterpretation and PlanAction, and the class Storage. This leaves only the implementing classes to be added. Observable variables from the ASDM are added as classes implementing Sensor, derivable variables implement Concept, and controllable variables implement Actuator.

If no controllable variables are provided, we assume a user interface, as was the case in the example described earlier.

The deduction of the application behavior can likely be partially performed. The causal links allow for reasoning about needed application behavior, so part of the DataInterpretation behavior may be deduced. However, as the planning and presentation of actions can not be captured in the ASDM, this has to be described by hand.

For the prototype implementation step, it will be possible to automate part of the steps. As the static system structure and behavior have been modeled in the previous steps, this information can be combined and code can be generated. This code is then to be extended with, among others, user interface code to complete the prototype implementation.

The automation of the process steps will improve the development speed of the applications, as well as reducing the number of errors made while designing the prototype.

V. RELATED WORK

[9] describes a tool that is to support knowledge workers to improve well-being at work. The application is to run on a personal computer system and use sensors in order to recognize the user’s context. The information sources proposed are the keyboard and mouse, but also application information. Using these, it can be decided what task the user is working on. By incorporating sensors such as microphones, GPS sensors and cameras, the user’s social context can also be deducted. A combination of these can tell us something about the perceived workload. Specifics regarding interventions if the derived workload is too high are not discussed.

The authors of [1] recognize that requirements engineering for context-aware applications is hard to do at design time. They propose to use a mobile requirements engineering
process in which a requirements engineer, who is supported by tools, follows the user during the testing phase of the application. If the user has specific requirements, these are recorded, together with characteristics of the current context. These requirements are then used to improve the application in an iterative way.

[2] discusses a method to improve the requirements engineering process for pervasive health care systems. Using a three-tier process, the authors discuss how to go from prose, to executable models, and finally to an animated application users can interact with in order to elicit better requirements. The primary difference with our work, is that the authors regard a system that is to be embedded in a fixed context, it staying stationary, whereas the applications considered by us are mobile; they are to be used in a multiplicity of places and contexts.

VI. CONCLUSIONS AND FUTURE WORK

The concept of context-aware applications that provide services tailored to the context at hand are highly promising. The design and implementation of these applications is however very challenging. Because users and developers alike have little experience with this applications that exhibit adaptive behavior and users have a hard time imagining every possible future context the application has to adapt to, traditional design and requirements engineering methods are unsuitable. We propose to use causal domain models to capture medical knowledge related to well-being. These models are then to guide the development process of the context-aware well-being application, allowing for causal reasoning over the variables relevant in the well-being domain.

The development process of a prototype starts with the full domain model being reduced to an application specific domain model that only captures the part of the domain that is of interest to the application under development. Using this ASDM, the static structure of the application is described, after which the application behavior can be modeled. Combining these two and using the ASDM, a prototype can be designed and implemented.

For future work, we propose to improve the development process by incorporating MDE tools and techniques to allow for partial automation. Furthermore, we want to improve the overall domain model by consulting more experts.

ACKNOWLEDGEMENTS

This publication was supported by the Dutch national program COMMIT (project P7 SWELL).

REFERENCES