An Engineering Approach to Adaptation and Calibration

Michael Fahrmair and Wassiou Sitou

Technische Universität München, Department of Informatics,
Boltzmannstr.3, D-85748 Garching (Munich), Germany
fahrmair@in.tum.de, sitou@in.tum.de

Abstract. A new computing era after Mainframes, PC's and mobiles is becoming more and more anticipated since the beginning of the 21st century. This new era is often described with several confusing terms such as pervasive, ubiquitous, ambient or context-aware computing. However, there is a common characteristic behind all these projections: They are all based on a substantially more flexible system understanding, whereby the thought of the system as a tool moves into the background and the needs and wishes of the user step into the foreground. Such concepts for software applications being aware of their context are in fact not new, but become more and more important nowadays for productive fields of software and systems engineering and particularly in ubiquitous and wearable computing.

1 Introduction

Adaptation in a common sense is defined as an act of changing (structure, form, or habits) to fit different environmental conditions [10]. For technical systems, such environmental conditions are usually referred to as *context* ([4], [8]).

Nowadays, adaptation needs to be considered as a key requirement for future mobile and ubiquitous systems that envision heterogeneous environments where system and application functionality needs to be dynamically adapted to constantly changing situations. There are existing architectures and frameworks such as [2] that support developing context aware software. However, the important aspect of designing context and adaptation logic is typically overlooked. The context model and adaptation decision logic are usually static and hardcoded in the adaptable entities. In view of ubiquitous systems, this approach seems inadequate, since the circumstances in which a system's functionality may be executed and the context parameters that may influence it, will not always be predictable a priori at the time the function is being developed.

In this paper we first describe what should be understood by adaptation and why this concept needs a particular consideration in software engineering research. After this, we introduce a generic, reusable mechanism that offers runtime customizable context criteria and adaptation algorithms.

2 The Scope of Adaptation

The main goal of adaptation is to achieve ubiquity. Ubiquity means enhancing usability of functionality in as many situations as possible. To be ubiquitous, functionality must meet three essential conditions. The necessary HW/SW infrastructure can be made available in the given situation (*availability*), the current user requirements (*applicability*) can be fulfilled and the necessary interactions should not be conflicting with the user's situation, i.e. his current free interaction possibilities (*operability*). It is obvious that a maximum ubiquity cannot be achieved by stacking up implementations for predefined requirements into a system limited by available resources. Moreover these limitations can vary from situation to situation. The resource limit is therefore usually given by tradeoffs between the least common denominator of providable functionality and a restriction of situations resulting in reduced ubiquity (*availability problem*). Besides that, additional interactions are necessary to communicate the current user's dynamic requirements selection, so that they can be mapped onto the superposition of static requirements fulfilled by the system. The amount and complexity of interactions between a user and a system is, however, also often limited by the current situation. Without adaptation, there are obvious tradeoffs between type, amount and complexity of user interactions and the number of functionalities a system can provide (*usability problem*).

Introducing *reconfiguration* solves the availability problem due to the fact that static requirements can be clustered into groups of not conflicting dynamic requirements (*configurations*) that are usually required by the user only in a certain situation. A system is *reconfigurable* if it supports changing between different sets of such configurations according to its situation. While it is possible to have reconfiguration directly controlled by the users, this kind of solution does not necessarily add to ubiquity because it adds additional user interactions in deciding about and choosing the proper configuration. To solve the usability problem of ubiquity therefore all or part of the manual reconfiguration management can be replaced with further functionalities fulfilling additional requirements of automatically reconfiguring between dynamic requirement implementations. For this issue, the system needs to decide on behalf of the user entity about its current valid requirements. This can be done by monitoring the system's environment using context. A *context* is the sufficiently exact characterization of a system's situation by means of perceivable information that is relevant for the adaptation of the system. It is a model of a situation containing all necessary and available information to reason about the user's or any other involved stakeholder's requirements. The overall process of adaptation therefore can be viewed as a three-level process: monitoring the context of a system, choosing the best appropriate configuration and deploying a new configuration at runtime. The deployment can involve changing behavior, implementations, modifying structure, adding or removing functionality or even downloading new code.

A simple context can be modeled using an entity relationship data model that holds the contextual information. The model proposed here is more detailed and differentiates in sensors, context data and interpreters as suggested in

[4]. Enriching context with its dynamic processing information (e.g. sources) has the advantage that new sensors and interpreters can be discovered and bound at runtime. The context model in [4] is however directly consumed by the context-aware system. Thus the adaptation logic and some context dependencies are hardcoded into the adaptive system. Hence the logic and results can not be shared among several subsystems and moreover the adaptation can not be reconfigured to meet varying resources. The model proposed here therefore adds not only sources and computational nodes, but also sinks for contextual information. *Actuators* represent parts of the system that access or observe parts of the context [7]. With this extension, adaptation can be defined as special interpreters that take information from initial or intermediate context and compute a specification of how the new system should look after the adaptation (*reconfiguration context*).

3 The Approach of Calibration

Realizing adaptation of a system using context information handling and decision logic that were designed at development time works quite well for small and specialized adaptive applications within relatively stable and predictable environments. Some context aware applications are also especially tolerant towards wrong adaptation by nature, because their results can be easily ignored, if wrong, while still deemed helpful by the user, if right[5]. Own experiments with adaptive systems confirmed that complex adaptive systems are usually supposed to fall into the trap of the frame problem [9], [3]. This is a well known problem from AI about the difficulties describing an infinite complex and dynamically changing world using static assumptions (i.e. models). Over time some of these assumptions and therefore abstractions used in a model can get wrong, even if they were valid while constructing the model. This leads to false (compared to reality) decisions [12]. Even if this problem is not generally solvable, it can be avoided or circumvented by changing the model that reasons about reality (i.e. our context adaptation) from time to time to meet the reality. This process is called *calibration* and can be seen as an adaptation of the adaptation. With state of the art adaptation and context-awareness concepts, there is however always an unchangeable part of a model like the logical service architecture, decision logic and context management.

Our approach therefore is mainly focused on reconfiguration of the adaptation logic; which can then be target of context dependent adaptation decisions. To achieve such a total reconfigurability we regard a description of the adaptation model as contextual information that can be modified by sensors and interpreters and then used by actuators to reconfigure the adaptation and context of a system (e.g. by adding a new context or decision logic at runtime). This is possible, because our model of the adaptation process was completed by introducing actuators (see previous section). Adaptation this way, like any other usage of context information, becomes visible, detectable and replaceable at runtime. A model of context adaptation that can describe its self reconfig-

uration is called calibrateable model (*k-model*). Any specific implementation of a k-model with an actuator that can read and reconfigure context adaptation models can serve as a generic framework, because it can be fed with any other specific specification of a context adaptation and still will reconfigure itself accordingly due to its total reconfiguration ability (for an example framework see [5]). However, a common formal founded semantic of the k-model is necessary to ensure that every refinement of the k-model remains consistent. A mathematical founded base model which consists of components and channels describing functions processing sets of infinite message streams [1] is used. Their behavior is specified as a relation between communication histories of input and output channels. Adaptation in this basic formal model is interpreted as a change of network components representing the adapted system. In the case of the formal base model, this can be defined as schematics mathematically describing the relation between communication histories of a set of typed input and output channels that can filter the output of certain components or channels that are not active in a certain adaptation state. Looking at the formalization presented in [5] it becomes clear that context adaptation is a purely engineering construct. Structuring certain behavior changes in a system the way it is expressed within the k-model and its formal foundation has clear advantages from a systems engineering point of view dealing with system flexibility as described in [5].

Seeing adaptation as an engineering construct only makes sense in conjunction with a technical possibility to automatically implement a specified system. In software engineering there are several concepts for dynamically implementing a software system such as late binding, DLLs and services. The most flexible concept to date however, are services that even allow for changing an active implementation at runtime (Design@Runtime [6]). Services are an sufficient technical concept to implement our mathematical model of adaptation since services usually are realized using a proxy access component that can act as a switch between several component implementations and therefore acts like the adaptation filter component (actuator) of our abstract model. However changing the system by switching component implementations has some invariants in form of the logical architecture (services), i.e. the fulfilled function, task or requirements of the given system or subsystem. To achieve the total reconfigurability necessary for adaptation calibration, a second switching/filtering layer needs to be introduced that can control the service proxies. Further details about this can be found in [6]. A short summary of the technical realization of the framework, the mathematical model and its formal description method was presented in [11].

4 Conclusion and Future Works

So far, context models for adaptability were incomplete and neither flexible nor modular enough. Incomplete because they mainly emphasize deriving context data from a system situation and disregard the opposite direction of adapting the system situation based on the context data. Not flexible enough, because context is defined regarding a system situation but the exact relation between

situations and context remains unclear [4], [9]. Our concept of an adaptation context therefore can hold a description of changes in the specification of context dependent adaptation behavior of the system itself. The process of changing context or adaptive behavior depending on the current context is not necessarily self-contained since the information can be produced by sensors and interpreters outside of the system [5], [11]. This way the system's context and adaptation behavior can also be modified by means outside the scope and knowledge of the system architect during design time of the system, thus providing enough flexibility to integrate further modular techniques (including human customization) to address the frame problem.

Further research effort needs to be put into applying and integrating various concrete approaches from AI research for changing adaptive behavior and relevance. What is also still missing is a semantically well defined process to design adaptive systems, while concentrating on the adaptive behavior rather than discussing implementation details.

References

1. Broy, M., Stølen, K.: Specification and Development of Interactive Systems - Focus on Streams, Interfaces and Refinement. Monographs in Computer Science, Springer-Verlag, 2000.
2. Chan, A., Chuang, S.: MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing. IEEE Transactions on Software Engineering, 29(12), 2003.
3. Dennett, D.: Cognitive Wheels: The Frame Problem of AI. In C. Hookway, editor, Minds, machines, and evolution, pages 129-151. Cambridge University Press, Cambridge, 1984.
4. Dey, A.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology, 2000.
5. Fahrmair, M.: Kalibrierbare Kontextadaption für Ubiquitous Computing. Dissertation, Technische Universität München, 2005.
6. Fahrmair, M., Salzmann, C., and Schoenmakers, M.: A Reflection Based Tool for Observing JINI Services. In Reflection and Software Engineering, number 1826 in LNCS. Springer-Verlag, 2000.
7. Houssos, N., Alonistioti, A., Merkakos, L., Mohyeldin, E., Dillinger. M., Fahrmair, M., Schoenmakers, M.: Advanced Adaptability and Profile Management Framework for the Support of Flexible Mobile Service Provision. IEEE Wireless Communications Mag, 2003.
8. Lieberman, H., Selker, T.: Out of Context: Computer Systems that Adapt to, and Learn from, Context. IBM Systems Journal, 39(3-4): 617-632, 2000.
9. Lueg, C.: Operationalizing Context in Context-Aware Artifacts: Benefits and Pitfalls. Informing Science, 5(2), 2002.
10. Merriam-Webster: Collegiate Dictionary. Merriam-Webster, Inc., 2003.
11. Mohyeldin, E., Dillinger, M., Fahrmair, M., Sitou, W., Dornbusch, P.: A Generic Framework for Negotiations and Trading in Context Aware Radio. In Software Defined Radio Technical Conference, Phoenix Arizona USA, 2004.
12. Pfeifer, R., Rademakers, P.: Situated Adaptive Design: Toward a Methodology for Knowledge Systems Development. In W. Brauer, D. Hernandez, editors, Proceedings of the Conference on Distributed Artificial Intelligence and Cooperative Work, pages 53-64. Springer Verlag, 1991.