A Generic Framework for Negotiations and Trading in Context Aware Radio

E. Mohyeldin[1], M. Dillinger[1], M. Fahrmair[2], W. Sitou[2] and P. Dornbusch[2]
[1]Siemens AG, St. Martin Str. 76, D-81451 Munich
[2]Technocal University of Munich, Boltzmann str.3, D-85748 Garching, Germany
eiman.mohyeldin@siemens.com
fahrmair@informatik.tu-muenchen.de

ABSTRACT

The support of reconfiguration requires the existence of negotiation procedures and trading rules connecting the device reconfiguration manager with the corresponding network proxies that will operate between the network proxies and the managed equipment, and will enable the exchange of information (e.g. mode availability, access network capabilities) or the ordering of certain procedures to take place. The task of the reconfiguration is to gather all the required software modules and carry out the individual reconfiguration steps.
However, in the future mobile terminals will have access to different radio access technologies and utilize different communication protocols depending on the requested application QoS. Moreover context aware application should be developed that can adapt their level of functionality to dynamic radio resource restrictions like available bandwidth, delay and link interruptions. Therefore a generic software framework for adaptation and reconfiguration is a necessity.
In this paper we propose a framework that supports the development of customized middleware, reconfigurable protocol stacks and adaptive application services in the three main layers of reconfiguration (infrastructure, application and user layers). The framework is composed of the following main phases: Profile and context management, Adaptations decision and trading rules and Realizations of the technical reconfiguration.
Keywords: ubiquitous computing, context aware, mobile service provision, 4G, adaptability, reconfigurability

1 Introduction

Adaptation is 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 like roaming across different radio access technologies, different device capabilities and user personalization needs. There are already several different architectures and frameworks supporting developed context aware HW/SW-systems (for a good overview see [3]). However, the important aspect of designing contexts and adaptation logic itself is typically overlooked. Moreover the context model and adaptation decision logic are usually static and hard-coded into the adaptable entities that are therefore suitable only for implementing relatively small scenarios within predictable environments. In view of 4G and ubiquitous systems, though, this approach seems inadequate, since the environment in which a system's functionality can be executed and the context parameters that may influence it, will not be predictable a priori at the time that the function is being developed. Generic, re-usable mechanisms that offer runtime customizable context criteria and adaptation algorithms therefore are a basic requirement for developing long-living ad-hoc reconfigurable protocol stacks or adaptive application services.
In this contribution such a generic software framework for adaptation and reconfiguration is introduced. The paper is organized as follows: in section 2 the process model that allows the design of context awareness is sketched out, concentrating on the adaptive behavior rather than discussing implementation details. Section 3 presents an integrated abstract model for adaptability that is based on mathematical streams and can be used to formally specify and design context aware systems taking into account both aspects of collecting context information and processing it. This abstract mathematical model then is implemented into a technical model which, combined with several helper components and a graphical description technique forms a generic formal founded framework with precise semantics in section 4. Section 5 concludes the paper with some recommendations and outlook for future research work.

2 Abstract Model for Context awareness

Context in an abstracted view is any information that characterizes a system's situation and hence equivalents to the information stored e.g. in a user, terminal or network profile. Since context is usually defined using an abstract view of a situation [1], a specific context is always specified from a certain perspective and describes only the relevant information from the system's environment. For example the user profile could be used as a context describing user preferences [2]. A user profile that is used in a terminal would most likely describe the user profile from the user's perspective of operating his terminal, including his preferences. The user profile that is used in a network management entity would most likely describe the user from

the network point of view, especially his current status, actions, subscriptions and billing information's.

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 proposed in [3] to enrich the static data structure of context with its dynamic processing information. *Sensors* observe the external system environment. They gather information that describes the system's situation and their changes and update the *sensor context* data according to the system situation. *Intermediate context* data in contrast is updated by *interpreters* observing the sensor context. Interpreters can calculate any information that can not directly be measured with sensors thus forming an abstract (interpreted) description of the initial physical (sensored) situation. A change of an intermediate context can of course trigger other interpreters, resulting in further context data changes and so on.

Since such a context model not only describes contextual information but also its sources it has the advantage that new sensors and interpreters can be discovered and bound at runtime. The context model itself however is directly consumed by the context aware system, e.g. for deciding about a certain reconfiguration based on a specific context state. Thus the adaptation logic and with it certain context dependencies are hard coded into the adaptive system, the logic and results can not be shared among several subsystems and moreover the adaptation can not be reconfigured itself (e.g. to meet varying CPU resources).

The model proposed here therefore extends the context model as defined in [3] by adding not only sources and computational nodes but also sinks for contextual information. Adaptation like any other usage of context information becomes visible (and detectable and replaceable at runtime) as *Actuator* elements. Actuators represent parts of the system that access or observe certain parts of the context.

Additionally it is also possible to move adaptation logic into such a context adaptation model instead of being hidden inside the actuators if the result of an adaptation decision is just modeled as another context describing how the system (architecture, functions, behavior etc.) should look like after the necessary reconfigurations will have taken place (*adaptation context*). With this extension adaptation can be defined as special interpreters that take information from the initial or intermediate context data elements and compute a specification of how the new system should look like after the adaptation.

The complete model for context awareness in this paper thus is made up of all initial, intermediate contexts and the adaptation context, the sources for information in form of sensors and interpreters and at least one actuator that reconfigures the system according to the description found in the adaptation context data.

3 Mathematical Model and formal description

Even the adaptation model itself can be such a description that is modified by sensors and interpreters and then used by an actuator to reconfigure the adaptation and context of a system (e.g. by adding a new context or decision logic at runtime). A model of context adaptation that can describe its self-reconfiguration is called calibratable 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 will reconfigure itself accordingly.

However a common formal founded semantic of the abstract adaptation model is necessary. Moreover a precise semantic is the basis for several important software engineering verification techniques like automated testing, model checking and theorem proofing.

A mathematical founded base model which consists of components and channels describing mathematical functions processing sets of infinite message streams[4][5] is used. In this base model; systems or subsystems are described as a network of components that communicate with each other over channels. Their behavior is specified as a relation between communication histories of input and output channels. A communication history is expressed as a stream of messages. A stream $s_n =< m_1, m_2, ... >$ is a finite or infinite sequence of messages $m_i \in M$. M^* is the set of finite, M^∞ the set of infinite message sequences. $M^\omega \overset{def}{=} (M^*)^\omega$ is the set of timed message sequences described as infinite Sequence of finite sequences over M .

The relation between the input (I) and output (O) message histories

$$F \in \vec{I} \to \wp(\vec{O})$$

describes a specification of visible behavior.

Adaptation in this basic formal model is interpreted as a change of network components representing the adapted system, i.e. components or channels can be added or removed resulting in a change of visible behavior of the system. However formalizing these kinds of dynamic changes is complex. Since in principle any model results from abstraction which is assumed static or at least its relevance is not changeable. Hence models are static approximations of reality but using static assumptions, dynamic can at best be emulated. Adaptation therefore can only be formalized using a superposition of all possible structures and functionality as well as a behavior to emulate

a system reconfiguration by switching back and forth between the adaptation possibilities. One could argue that it is therefore not possible to formalize adaptation because it is also sometimes seen as a method to reconfigure a system into a state that is unknown at specification time of the system, e.g. by downloading new software modules at runtime.

However there is a slight but important difference between the specification (model) of a system and its implementation. While the implementation is part of reality, a specification is a model of this and maybe similar realities by restricting a state of all possible systems down to a usually still infinite group of systems showing the expected behavior (the static approximation mentioned above). Adaptation can thus easily be modeled as a superposition of states without needing to enumerate or even know each and every possible adaptation state as long as the switching behavior between them can be expressed. 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 (see Figure 1 the D0 component filtering out one of two alternative configurations).

Looking at the formalization it becomes clear that apparently context adaptation is a purely engineering construct; since extra functionality is not added (we still only have channels and components). Structuring certain behavior changes in a system the way it is expressed within the abstract context adaptation model and its formal foundation, however has some clear advantages from a systems engineering point of view dealing with system flexibility:

- A clear segmentation and precise switching mechanism allowing for running a system even if it is only partially implemented. So not all parts of the superposition need to be implemented at the beginning and can be loaded afterwards. Precise adaptation makes sure no specified behavior is exposed that is not implemented yet.
- Decoupling communication between certain components such as sensors using the context allows for communication of components with implementations that not even exist at the same time or which have an availability (like in wireless or mobile networks) that can not be controlled by the system itself. So even if a sensor for needed information is not available at the moment, its last known value still might be available from the context.
- Communication API defined for sensor-, interpreter-, actuator- and context components ensuring that all current and future components can establish a very basic communication with each other (i.e. exchanging context data. Based on this, enhanced protocols can be negotiated. This allows for easy expansion of a system with new components and functionalities that were unknown to the developer of the system.

4 TECHNICAL FRAMEWORK

Several concepts for dynamically implementing/extending systems are existing. The most well known concept is dynamic (late) binding of methods or functions within the scope of object oriented inheritance or dynamically linked libraries (DLLs) that are loaded and bound on demand. The most flexible concept to date however are web services that allow for changing an active implementation at runtime. The service itself is specified as a group of similar behavior technically represented by a transparent access proxy, while abstracting component (implementation) identities. Services therefore can be understood as a logical architecture or network of static interdependencies between all possible implementations that could be used at runtime to implement a given task.

Since more than one component implementation can be used to implement a given service or one component implementation can be used to implement more than one service, this process (of binding an implementation to a service) is sometimes referred to as Design@Runtime [10].

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 our 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. Restricting adaptation to the use of services therefore only produces a partial reconfigurable system.

Realizing total reconfigurability of a system using adaptation require the extension of the service concept such that the logical architecture, i.e. function, task or requirements are changed. This is especially important for complex adaptive systems, since they are very likely to fall into the trap of the frame problem [1][6] . In short this is a well known problem from KI about the difficulties describing an infinite complex and dynamically changing world using static assumptions (i.e. models). Therefore over time some of these assumptions and therefore abstractions used in a model can get wrong even if they were valid at the time a model was constructed. This again leads to false (compared with reality) decisions [7] like an intelligent fridge that can not know that the expensive food stored in it

is only used for tomorrows special occasion party and needs not be ordered again if used up.
Even if the problem is not 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 fit a reality that might have changed. This process is called calibration and can be seen as an adaptation of the adaptation itself. However it is obvious that with partial reconfiguration there is always part of a model that can not be changed (usually the service proxies that provide transparent access of varying components that can be bound at runtime).
The technical framework therefore relies on total reconfiguration that is achieved by extending the services concept with activators. The activator is itself a service that controls the reconfiguration channels of all possible service proxies in a given system. The activator can set the component implementation that is used by a service proxy or can switch off the service proxy by deactivating its output channels that are observable from the outside of the system. The activator also controls its own service proxy. This way it can hand over the reconfiguration control to any other activator component implementation achieving a total reconfigurability without any invariants if necessary, see Figure 1.
The framework is completed by an implementation of the context and activator component as well as an adaptation model sensor. This way any systems adaptation behavior can be bootstrapped like sketched out at the beginning of section 3. A description of the concrete adaptation is loaded from an outside source into the context server using the adaptation model sensor (e.g. loading from a file). A special actuator called the model actuator reads the adaptation model description from the context. Since this model actuator is implemented as an activator it can activate and bind any further sensor, interpreter, actuator and context service that is described in the adaptation model description loading at bootstrap time.
Also at any later time it is possible to modify (calibrate) this adaptation description stored in its own context again as long as the components defining the bootstrap adaptation are still present and active or were replaced by implementations of the same functionality.

4.1 Specification and description techniques

The model of a concrete adaptation behavior stored in its own adaptation context is described as an XML document. This document contains information about all sensors, interpreters, context elements and actuators that are described as services with their IDs, syntactical and semantically type information. The syntactical type is usually composed from an interface description (IDL, WSDL etc.) but can also contain binding information like the reference of a component instance. Semantic type information in contrast can be used for ontology based searches without specifying a concrete interface. This is true since the abstract model consist of only four basic roles with minimum communication APIs, therefore it is possible for example to search and bind two components with unknown interfaces. It just needs to be made sure that the two can communicate with each other. Being one of the four adaptation model element roles ensures automatically that the roles can be embedded and used in the adaptation process even if the functionality and interface protocol is completely unknown at construction time of the initial reconfigurable system.
The XML document used for describing flexible adaptation models can moreover be easily mapped onto the formally founded mathematical model. Allowing for a wide range of runtime checks of such specifications, for example testing for consistency before deploying a new adaptation behavior.
The framework contains as well a syntactical mapping between the texts based XML description and a graphical notation, see Figure 2. The graphical specification technique is utilized for designing support tools for developing adaptive applications.
The given framework furthermore contains a set of specialized syntactical transformers that can modify the original graphical specification resulting in a refined and clear design specification, for example by automatically generating indexed sequentializations of complex models or information folding/unfolding techniques.
This transformators can also be used to generate descriptions of adaptation behavior that are understandable by end users, e.g. in the form of device manuals, online help or editors that allow the user a limited personalization of adaptation behavior.

5 Conclusions and outlook

The framework presented in this paper is a generic approach to support all kinds of adaptation in reconfigurable SDR systems. With its support for calibration even the adaptation logic itself can reconfigured to avoid typical framing problems like spontaneous unexpected behavior that can emerge especially in long running systems or consumer systems with a large number of users with different and changing expectations toward a semi-intelligent system.
Besides an environment that allows for changing adaptation behavior at runtime the framework makes use of a formally founded abstract adaptation model to allow for sophisticated and fully automatic tool supported specification, deployment and documentation of adaptation behavior. All methods and techniques can be customized according to experience level of the user even allowing for runtime personalization of adaptation behavior by the end user.

Since the framework supports generic adaptation such a personalization mechanism is not limited to modifying a simple set of rules. Instead it is possible to rearrange abstract function roles (sensors, interpreters, context and actuators) that can hide any kind of technical realization. Therefore it is even possible to mix rule based decisions with fuzzy logic components or neuronal networks to customize an adaptation behavior.
Despite this high level of flexibility applications based on this framework, like the wireless middleware scenario demonstrator described in [9], the performance and scalability is far better than comparable middleware approaches [8].

6 References

[1] Christopher Lueg. Operationalizing Context in Context-Aware Artifacts: Benefits and Pitfalls, Informing Science Vol. 5 No. 2/2002. http://informingscience.com/Articles/Vol5/v5n2p043-047.pdf.

[2] H. Peter Alesso and Craig F. Smith. *The Intelligent Wireless Web*. Indianapolis: Pearson Education, 2002.

[3] Anind K. Dey. Providing Architectural Support for Building Context-Aware Applications PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[4] Manfred Broy. Compositional Refinement of Interactive Systems Modelled by Relations. In W.-P. de Roever, H. Langmaak and A. Pnueli, editors, Compositionality: The Significant Difference, number 1536 in LNCS, pages 130-149. Springer Verlag, 1998

[5] Manfred Broy and Ketil Stolen. Specification and Development of Interactive Systems - Focus on Streams, Interfaces and Refinement. Monographs in Computer Science. Springer Verlag 2000

[6] Daniel C. Dennett. Cognitive wheels: The frame problem of ai. In C. Hookway, editor, Minds, machines, and evolution, pages 129--151. Cambridge University Press, Cambridge, 1984

[7] Pfeifer, R. and Rademakers, P. (1991). Situated Adaptive Design: Toward a Methodology for Knowledge Systems Development. In Brauer, W. and Hernandez, D., editors, Proceedings of the Conference on Distributed Artificial Intelligence and Cooperative Work, pages 53-64. Springer-Verlag

[8] A. Chan, S. Chuang. MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing. IEEE Transactions on Software Engineering, Vol. 29, No. 12, December 2003

[9] DuReau P., Redmill D., Mohyeldin E.; Golubicic Z.; Hirschfeld R.; Fahrmair M.; Salzmann C.; Dornbusch P., "Description and Specification of the SCOUT hardware validators and SCOUT middleware demonstrator" IST-2001-34091 D4.3.1, 2003-11-02.

[10] M. Fahrmair, C. Salzmann, and M. Schoenmakers. A reflection based tool for observing jini services. In Reflection and Software Engineering, number 1826 in LNCS. Springer Verlag, 2000.

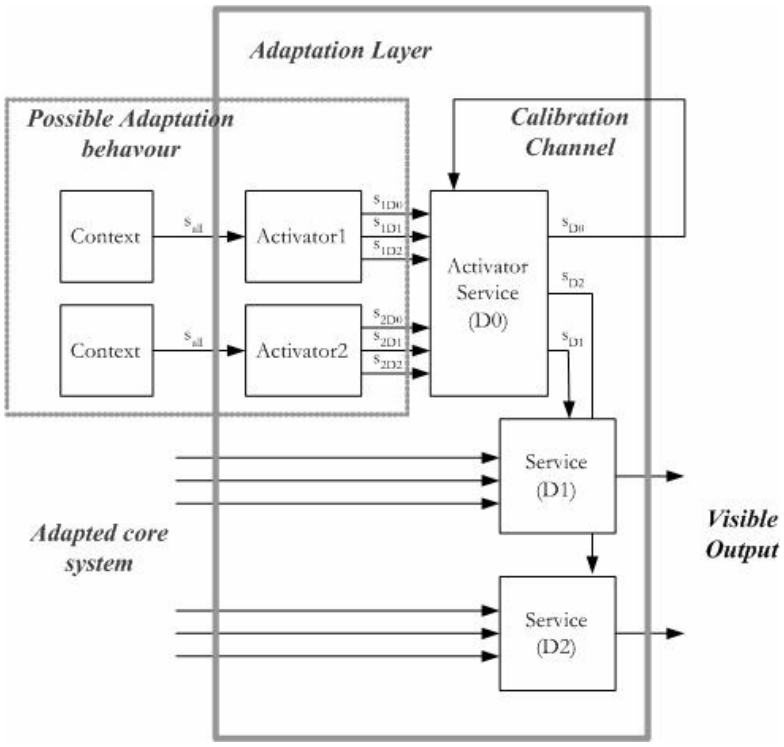

Figure 1: Adaptations schematics

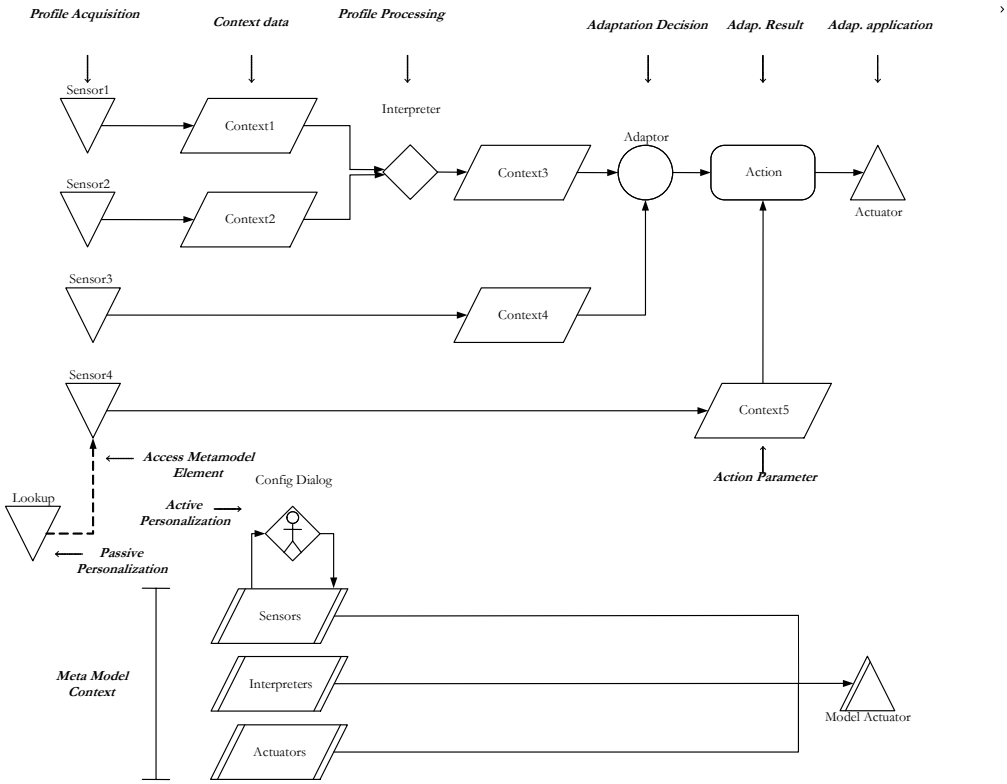

Figure 2: Graphical Notation for Adaptations Model