T U M

I N S T I T U T F Ü R I N F O R M A T I K

CAWAR - Formalizing a Framework for Ubiquitous Computing Applications

Michael Fahrmair, Christian Leuxner, Wassiou Sitou, Bernd Spanfelner

TUM-I0830
August 08

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-08-I0830-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

CAWAR – Formalizing a Framework for Ubiquitous Computing Applications

Michael Fahrmair[1],
Christian Leuxner[2], Wassiou Sitou[2] and Bernd Spanfelner[2]

[1] DoCoMo Communications Laboratories Europe GmbH
Landsbergerstr. 312, D-80687 Munich, Germany
fahrmair@docomolab-euro.com

[2] Technische Universität München, Departement of Informatics
Boltzmannstr. 3, D-85748 Garching, Germany
{leuxner|sitou|spanfeln}@in.tum.de

August 21, 2008

Abstract

The concept of *Ubiquitous Computing*, also called *Ubicomp* for brevity, describes a new paradigm on the usage of computer based systems. As opposed to most "conventional" computer based systems, Ubicomp applications are characterized by an enhanced usability, thus enabling users to benefit from computer usage and support in as many situations as possible. The computer as an apparent tool steps into the background, while the actual needs and wishes of the current user step into the foreground.

The notion of context awareness – and related terms such as context adaptation or adaptation for short – constitute enabling technologies for realizing computer systems according to the Ubicomp paradigm. Since usability issues are at the heart of Ubicomp applications, the adaptive systems try to gather information of their environment in order to automatically infer the wishes and needs of a user. Assumptions concerning these needs are in turn used to trigger system reactions on behalf of the user. This adaptation mechanism can be seen as an additional system feature, which raises the complexity of the overall system. Since the adaptation essentially relies on assumptions concerning the user and its environment, we suggest a special methodological and technical treatment of these aspects within the development process.

In this work, we mainly focus on the technical aspects of the topic and introduce a framework for realizing Ubicomp applications. The framework relies on a so called adaptation model, which describes the adaptive system behavior of the application. The adaptation model, or K-Model for short, *explicitly* expresses the assumptions concerning the user and the system environment. We investigate the intuitions before providing formal definitions of adaptation and related concepts, which serve as the formal foundations for the Cawar framework.

Contents

1 Introduction

Ubiquitous Computing is a paradigm for the usage of computer based systems, that combines a lot of aspects with a common direction towards a better usability of systems. The term was first used by Mark Weiser [Wei91] in 1991. Within this article, Weiser describes the future of computing as an accumulation of systems, which are distributed everywhere in our environment and collaborate in order to provide a user the most exciting computing experience. During years it became apparent, that this fiction includes a wide range of technical challenges. Obviously, there is a need to facilitate ad-hoc communication. This furthermore requires appropriate communication mechanisms as well as ontologies to allow for a common understanding of systems.

Beyond these technical questions, it soon has been recognized that applications could contribute to Weiser's vision as well. Applications that incorporate their usage context in order to relief users from excessive interactions have the potential of providing a better usability – even in situations where conventional computer based systems can not be used at all. Commonly, those applications are denoted as being context aware.

Basically, there is no concise definition for context aware applications. This circumstance is down to the fact, that an definite characteristic, which discriminates context aware applications from say reactive system, can not be found. Eventually, also context aware applications are systems reacting on inputs.

However, there is a methodological reason for taking the notion of context awareness into account. When considering context aware applications, people usually do not cite a trivially small system that switches on the lights if a motion is detected[1]. People rather cite applications that incorporate at least some complex situations of use, like driving a car or the accomplishment of a surgery. Systems being used in such situations, have two particularities that complicate their engineering: i) the detection of a complex situation like a certain phase within a surgery is not trivial and depends on a variety of influences within the operational environment. The system has to sense and interpret plenty of contextual information, which is prone to misinterpretation and errors. ii) in addition to the already complex system functionality an additional management functionality is required, which realizes the logic for deciding on the effects of some context conditions being detected. The design of this adaptation logic requires additional engineering efforts.

Within this document, we propose an approach for structuring behavioral aspects of context aware systems by means of a formal modeling technique. We introduce a technical framework, that enables to explicitly model the decision logic concerning the adaptive aspects of the system behavior. This structuring principle makes the overall system complexity more manageable due to separation of concerns. We exploit services for describing the system functionality and its dynamics along the dimension of context. The resulting system model

[1] although there is no good reason to disapprove with such an example

describes structural aspects, the communication and the workflow concerning the retrieval and processing of data, and also the interaction with the environment. We will give a detailed introduction to the so called K-Model, which is an approach to modeling the above mentioned aspects, and its formal foundation.

The remainder of this paper is structured as follows: in section 2 we informally introduce all relevant concepts related to the CAWAR framework. Notions such as context, adaptation and the architecture underlying the CAWAR framework are discussed. Afterwards, we define the introduced concepts by means of an appropriate formalism in section 3. We choose the FOCUS theory [BS01] for this purpose, because it provides practical modeling techniques for treating the aspects of system structure and interface behavior on a mathematical basis. Finally, section 4 draws a conclusion.

2 Informal definitions

Before going into the details of formalizing the CAWAR framework and its underlying concepts in section 3, some important on-topic concepts are introduced in the following. The generic definitions in this section are mandatory for the understanding of this paper and hopefully provide a common understanding of all considered terms. Most of the informal definitions are picked up and formalized by means of the FOCUS theory in the subsequent section 3.

2.1 Context

It's no wonder that the notion of context constitutes a very central issue in conjunction with context aware systems. We define context according to the following definition.

Definition

☞ Context denotes the sufficiently exact characterization of a system's application situation. The situation comprises all information, which is relevant for automatically adapting the system behavior. Moreover, the information must be *observable*, i.e. it is revealed by at least one interface

In other words, context constitutes an abstract model of the systems environment. For the success of context aware systems it is crucial, that this model is thoroughly engineered. We experienced that context aware systems are often equated with location based systems. The latter change their behavior according to their actual location or the location of the user. However, as argued in [SBHW99], there is more to context than location. Since the focus of this paper is on the formalization of the CAWAR framework – and not on the adequate engineering of a system's context – we refer the reader to [Sch02, FLSS08, SFS06, SS07] for further details concerning the elicitation and design of context.

2.2 Context adaptation

As indicated by figure 1, the concept of context adaptation can be understood as a process consisting of the four steps *context acquisition*, *situation identification*, *adaptation decision* and *adaptation realization*.

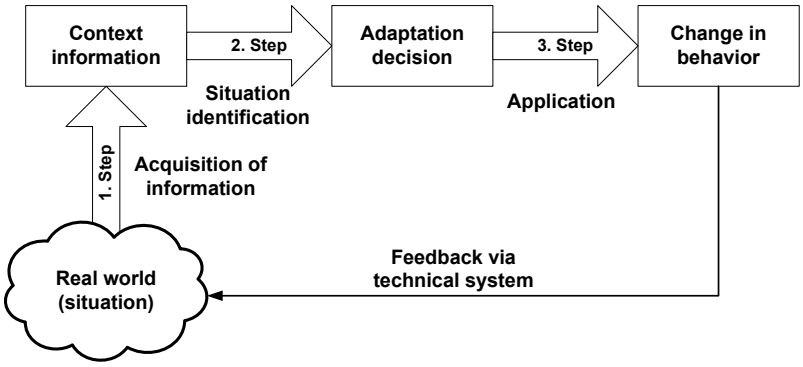


Fig. 1: Process of context adaptation

Our informal definition for the notion of context adaptation reads as follows:

Definition

☞ Context adaptation denotes the automated adjustment of the observable behavior of a system. Context constitutes the basis of decision-making for this adjustment. Adjusting a system's behavior is thereby always associated with changing its (global) state.

Automated in here means, that the adaptation itself is accomplished without any explicit user interactions. To emphasize this: it seems obvious that context adaptivity contributes very little without being executed automatically. The reason for this observation is, that – beside being able to recognize and evaluate different usage situations – context adaptive systems need to make the appropriate adaptation decisions *on their own*. A certain level of autonomy is necessary to satisfy the usability requirements typically imposed for those systems, namely to be usable in as many situations as possible.

The following example illustrates this requirement: consider a driver, which is currently unable to read an incoming message on his car's on-board display, e.g. due to a critical traffic situation. This driver is neither capable to navigate through a user menu in order to activate the speech output; the system is supposed to activate this function by itself – automated.

2.3 Context adaptive system

The definition of context adaptive systems given in the following, is sufficiently exact for the purpose of this paper. However, it does not provide a criterion for

definitely delimiting systems exposing an adaptive behavior from those which do not. We will investigate the notion of context adaptive system behavior in an upcoming publication. In short, this definition explains that adaptive behavior formally cannot be considered without referring to a subject (person or other technical system), which experiences the adaptivity by observing more or less non-deterministic system reactions. The degree of observed non-determinism depends on "how well" the subject knows the interface and the I/O behavior of the total system, i.e. whether the subject is able to observe (some of) the implicit system inputs received over sensors.

Since the purpose of this section is to establish a common understanding of related terms, we are content with the simplified definition given in the following.

Definition

☞ A context adaptive system (CAS) is an interactive system, that is able to observe the domain it shares an interface with. The CAS adjusts its structure or observable behavior in order to react to changes in this domain and to satisfy constraints imposed by those changes – even without the explicit interaction of a user. Therefore, context adaptive systems require an explicit model of the usage context (e.g. locations, participants, activities etc.), which comprise all domain aspects relevant for the system's adaptations.

Changing the system behavior is technically realized by altering, adding or removing certain software components of the system and/or their communication structure. For changing the system's behavior, we introduce the concept of a configuration, which we define as follows:

Definition

☞ A configuration denotes a set of interacting system components together with their communication structure. A configuration always exposes a certain I/O behavior.

Accordingly, changing a system's configuration means to switch from one configuration to another. This process is referred to as *reconfiguration*. In the case of context adaptive systems, this process is often controlled by a management layer realizing the adaptation logic of the system on basis of context information.

2.4 Calibration

We experienced that context adaptive systems are prone to a phenomenon called Unwanted Behavior (UB). A detailed discussion of this phenomenon can be found in [FSS06]. Since it originates from an inherent and potentially unsolvable problem of systems with a certain degree of autonomy (frame problem cf. [Den84]) we introduce a possibility to circumvent the effects of UB instead

of giving a solution to it. We call this approach calibration and define the term as follows:

Definition

☞ The calibration of a context adaptive system denotes the manual adaptation of the system's adaptation logic itself. The calibration is typically accomplished off-line, i.e. after an execution of the system, and requires some expertise concerning the logic of adaptation.

A prerequisite for calibrating an adaptive system is, that the adaptive behavior of the system is explicitly modeled – in our case by means of a *calibrateable adaptation model* or *K-Model* for short (cf. section 2.5). An explicit representation of the adaptation logic as a K-Model enables the adaptation of the model itself. Since deficiencies of a model in general can not be recognized from the model itself (cf. [Den84, Sim69]), a higher instance (e.g. a human) is necessary for analyzing this adaptation model. Consider this instance to be another technical system; this instance is prone to UB as well so that the problem is merely delegated to the next instance and so on. In consequence, we argue that the last instance eventually has to be a human being in order to enable an effective deficiency detection and handling, respectively. Clearly, the question "are humans subject to Unwanted Behavior?" is indeed philosophic and beyond the scope of this technical report.

In essence, calibration provides a possibility to correct system behavior from "outside" the considered system. The calibration might be accomplished after an UB occurred or as soon as an UB can be predicted due to knowledge about the adaptive behavior and upcoming situations, respectively. It seems obvious, that an understandable representation of the adaptive system behavior facilitates such an UB prediction. Calibration in our approach relies on a very special feature of the K-Model. The model contains a description of itself. It is used at runtime as a pattern to implement the specified system by finding and binding software components in accordance to the K-Model. The K-Model is also communicated to the user (or at least an assigned technician), which ideally identifies the model-parts responsible for the UB – and changes the adaptation model accordingly. The system afterwards reconfigures according to the changed specification and hopefully exposes the desired behavior. See [FSS05] for a detailed discussion of the calibration mechanism.

2.5 K-Model – calibrateable context adaptation model

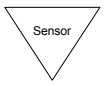
The K-Model constitutes a modeling technique for constructing the adaptation logic of context adaptive systems within a structured model. The basic elements of each K-Model are *services*. Since there are a lot of interpretations concerning the notion of services, we first of all clarify our understanding of the term within the context of this technical report. We use the term from a technical perspective: a service denotes a proxy, which exposes a potentially partial I/O

behavior and represents a set of service-fulfilling components with the same syntactic and semantic interface. We use services to achieve a loose coupling between software components. That is, an indirection is introduced between two interacting components by means of services: components only communicate via service proxies, which transparently forward the received messages to their service fulfilling components. In other words: services only describe behavior that is exhibited or required. Components are technical entities (hardware or software) that do the actual work to provide a service. This understanding is clearly related to the notion of web services.

The advantage of this additional abstraction layer is, that we only consider required system functionality, without having to deal with the realization of that functionality. This allows us to use different components as service realizations – depending on their availability or quality of service (QoS). On that abstraction level, we can flexibly add or remove functionality as required by the current usage situation.

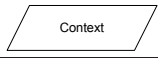
Basic elements

The K-Model makes use of four basic elements: *sensors*, *context elements*, *interpreters*, and *actuators*. By restricting their combination a structured description of any complex system behavior can be expressed. The elements themselves represent services with a predefined syntactic interface.



Sensors

Sensors are responsible for retrieving information from in- and outside the system. Therefore, they write sensed information to dedicated buffers called context elements. Sensors qualify to model physical sensors (e.g. thermometer, light sensor) as well as internal or external software entities (e.g. remote web services) or even human beings (terminal inputs can be regarded as context, too). The graphical representation of a sensor is a bottom-up triangle.



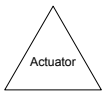
Context elements

Context elements are buffers for storing arbitrary information. They decouple any of the other three service types, since the latter are only allowed to communicate via context elements. Context elements represent uninterpreted sensor data as well as linked or interpreted information or even decisions. Their graphical representation is a parallelogram.

Interpreters

Interpreters are information processing entities. They read from context el-

ements and store their output to context elements. The corresponding I/O-transformation depends on the interpreter's logic and may be arbitrary complex (e.g. rule engines, neuronal networks) and can be used for decision making for prospective adaptations. Their graphical representation is a diamond.



Actuators

Actuators are responsible for realizing the system's adaptations by interacting with the core system and the environment, respectively. They read context elements, which contain information originating from interpreters and/or sensors, and subsequently execute the necessary adaptation decisions on basis of this information. Hence, their incoming context is also called "adaptation context". Their graphical representation is a rectangle.

The following definition outlines the main characteristics of a K-Model. The "activator" constitutes a special actuator within the CAWAR framework, which discovers and binds services to their service-fulfilling components at runtime.

Definition

☞ A K-Model denotes the model of a calibrateable context adaption. This model comprises an activator, which controls a set of services composed of sensors, interpreters, actuators and context elements [FSS05].

K-Model structure

We defined some modeling constraints for K-Models. A very central design decision is, that every *sensor*, *interpreter* and *actuator* is only allowed to communicate via context elements. This ensures a decoupling of the elements and guarantees local consistency of information, in case certain K-Model elements are exchanged or removed.

The K-Model describes the workflow of information processing within the adaptation logic. Sensors acquire the information necessary for adapting to the system environment. This information is stored as sensor context. Interpreters process (aggregate, accumulate) this information. Some interpreters only combine information or draw conclusions from information, that results in further/augmented information about the environment or the current situation. This context is called intermediate context. Finally, some interpreters render a decision concerning the possible system reactions and possible adaptations. This information is called adaptation context or adaptation result (cf. fig. 2).

The elements of a K-Model may be connected as described in the following: a unidirectional edge describes a flow of information between at least two elements. A sensor is only allowed to write to exactly one context element. Context elements are connected to at most one data source, but may be read by an arbitrary number of information sinks. Interpreters can handle any number of input

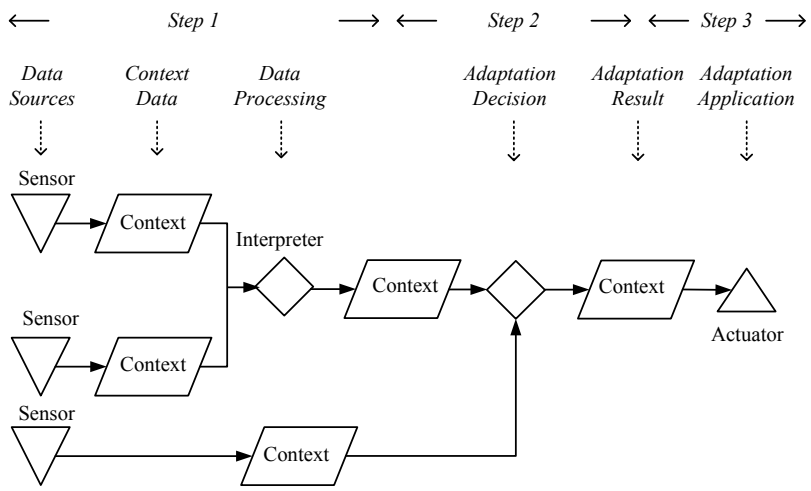


Fig. 2: Graphical notation and typical workflow of K-Models

and output connections and actuators can deal with any number of inputs. However, we recommend to delegate the preliminary processing to interpreters and to forward only one prepared context element to an actuator.

Dashed lines within K-Models have a special meaning. Each element has a so called *meta context*. This meta context describes the syntactic interface, quality os service and specified behavior of the element (syType) and its exact meaning (smType). It is crucial to understand the difference. The syType is sufficient to describe the exact behavior of the element (note that this embraces its behavior). The smType is used to distinguish technically identical entities. For instance consider two thermometers. Both may have the same syType. Nevertheless it is of fundamental difference if one is placed in a room or it measures the outside temperature. This difference is captured in the smType (further details are given in section 2.7). A dashed line describes a data flow to an element's meta context, not the element itself. Thus, by altering the element's description it is also possible to change its realization. We use this mechanism inter alia for controlling the reconfiguration. That is, we eventually change the set of service-fulfilling components, which should be bound to the element underlying the meta context.

2.6 Common architecture for adaptive systems

Although there exist a couple of approaches and frameworks for designing systems exposing an adaptive behavior, certain commonalities of the corresponding system architectures can be identified. Most architectures base on the idea, that the observable behavior of the system can be changed at runtime. However, we do not consider systems, which are able to "learn" any new behavioral patterns. We assume all behavioral variants, which the system might expose, to be specified at design time. In other words, the system is not able to change its overall behavior. It only changes the *currently observable part* of the former,

thus creating the illusion of dynamically changing its behavior. This process is technically realized by a filter, which mediates the communication between the system and its environment appropriately. To emphasize this: our architecture is based on the idea, that the system can not partake in any meaningful interaction with entities, whose interface it does not know in advance.

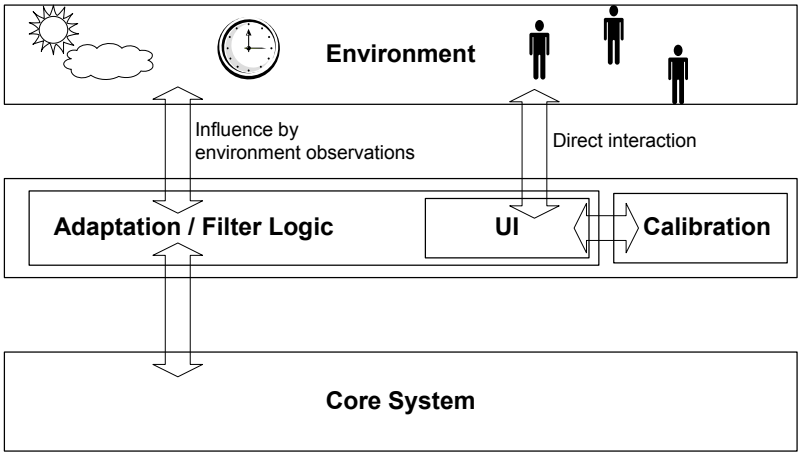


Fig. 3: Common adaptation architecture

Figure 3 illustrates the common architecture for context adaptive systems. The system core comprises the sum of all system behaviors. In order to guarantee the availability of some, the corresponding components must be installed on the main device the system is running on. Additional components may be found and bound in the environment as long as their syType and smType are known and can be matched to the system specification.

Essentially, the adaptation layer provides the filter for mediating the communication between the system and its environment. This filter is responsible for adapting the system behavior. All behavioral patterns the system might expose are specified within the adaptation layer by means of a K-Model. The layer can be decomposed into the following subsystems:

User Interface – UI The user interface is intended for any direct user interactions. That is, any information a user explicitly inputs to the system (e.g. press “send email” button) or any output that should be forwarded to the user (e.g. display email on screen) is mediated by the UI.

Adaptation/filter logic This subsystem comprises the logic for all possible adaptations the system is able to accomplish. The UI is part of this subsystem, because it may be subject to adaptations as well. The logic specifies, which part of the overall system behavior should be exposed under certain predefined circumstances (situations). The adaptation behavior is influenced by inputs via the UI and contextual inputs measured by sensors. The layer controls the observable system behavior by discovering and deploying the components within the system core and the proximate system environment.

Calibration The calibration enables a manual adaptation of the filter logic. This mechanism is necessary to circumvent the occurrence of Unwanted Behavior caused by the frame problem (cf. section 2.4). The calibration is eventually triggered off-line by experienced users or system administrators over a dedicated user interface.

The system environment comprises anything not subsumed within the core system and the adaptation layer, respectively. Prominent examples are physical conditions (e.g. weather, location), users and other participants as well proximate components (e.g. printers, screens) discovered at runtime.

2.7 CAWAR framework in a nutshell

The CAWAR framework is a generic approach to support all kinds of adaptation in reconfigurable systems. Selected aspects of this framework, which are necessary for the understanding of how context adaptation models are technically realized, are outlined in the following sections.

Framework overview

In this section a short overview about the overall CAWAR (Context AWare ARchitectures) framework is given, while the two subsequent sections discuss certain framework concepts in more detail. The framework principally consists of the following elements:

1. A set of *components* comprising the technical implementation of typical infrastructure functionality, i.e. context storage, discovery, etc.
2. A set of low level *interfaces* (API), that provide the most generic abstraction of context management, i.e. sensors produce context, actuators consume context, etc.
3. A *reference architecture* that suggests a basic generic pattern of how a context adaptive system can be designed in a completely reconfigurable way – formally, context adaptation can be understand as a self reconfiguring filter (cf. section 3.4). Following that pattern, any implementation of a context adaptive application can serve as a framework for bootstrapping any other context adaptive application.

Components, interfaces and architecture together form a basic framework for context awareness and adaptive applications. To develop a certain application, the framework merely must be fed with the desired specification of system behavior in form of a K-Model. Furthermore, the components fulfilling the respective services must be loosely combined with the framework according to the K-Model. Such components for example can also be detected at runtime.

The framework initialization is conducted by a designated actuator component named *model activator*, which expects a list of all required services (logical

service descriptions) as well as their corresponding components (technical realizations or references) from the context itself. Such a description is e.g. given by a XML file representing the adaptation behavior of the considered system, i.e. the K-Model. This model has to be previously read by a special sensor and written into an appropriate context element. The context comprising the K-Model can be further processed – allowing for self introspection and self adaptation – before it is ultimately deployed by the model activator. The latter finally reorganizes the services (sensors, interpreters, etc.) as well as their corresponding component bindings, thus reconfiguring the system in order to technically implement the K-Model.

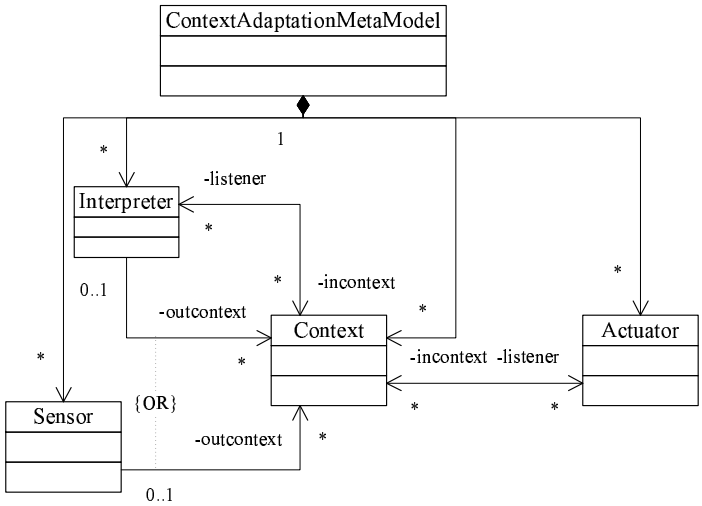


Fig. 4: Meta model describing the principle structure of the K-Model

The underlying meta model of any K-Model, which describes the principle composition of its logical service architecture, i.e. its grammar, is illustrated in figure 4. A concrete K-Model, containing the particular services which constitute the adaptive behavior of a considered application, is indeed an *instance* of the depicted meta model, which can be read by the model activator for initializing the context adaptive system as mentioned above.

Syntactical and semantical types

All services contained in a K-Model are specified by a logical service description (meta context). Each description thereby contains a syntactical (syType) and semantical (smType) description of its underlying service. Both types are used to match a suitable technical component that could implement the specified service. In other words, syType and smType specify, which components the service could possibly route messages to.

SyType describes any higher level protocol the service implementing component should understand – using the standardized low level interfaces of sensors, interpreters, actuators and context elements, including at least the data format

accepted (interface). Moreover, it could contain any other technical information needed to reduce the number of matching components, e.g. specification of behavior, QoS parameters, billing information etc. SyType contains information, that is needed by the activator in order to contact and bind possible candidates. An example constitutes a reference to a single component instance, which guarantees that this component is bound to the corresponding service.

SmType in contrast describes the meaning or usage intention of a certain component instance, besides its technical characteristics. This description is used to distinguish between several instances of technically identical components. Consider several identical temperature sensors or terminals, which are connected to the same adaptive system. Clearly, these devices might perform different "roles" with respect to the actual context, e.g. outside temperature, inside temperature, kitchen terminal or entrance terminal. An activator needs to distinguish, which component instance should be bound to a sensor, that feeds a context element with a meaning of "outside temperature" – expressed by that context element's smType. A syntactical description is insufficient in this case, since it could match more than one component instance. Instead, one of the available sensors needs to have been marked with a meaning of "outside temperature" as well. Semantical marking is part of the context and one of the main tasks of calibration.

It should be mentioned that the real "meaning" (smType) of a component is only generated by observation in a larger correlation with other entities, and can not be grounded in a symbolic description of the component instance alone. An indication of this fact is a component instance, that – though it has a constant behavior – can have different meanings in two different observation contexts. To give an example: the same camera instance, that shows i) the entrance of a building for one observer, could show in the same picture ii) a certain street segment for another observer, or iii) the weather conditions for a third observer, iv) the water level of the nearby river and so on. Another example constitutes a temperature sensor on the outside of a package. The sensor can be intended to measure the outside temperature (compared to the package's inside temperature), but at the same time measures the inside temperature for the owner of a storage house, in which the package is currently stored in.

Application subsystem

A context adaptive system built with the Cawar framework typically consists of three subsystems: i) the adaptation subsystem embracing all parts, which are subject to the frame problem and therefore need to remain totally reconfigurable, ii) the system environment containing all service fulfilling components, which are not permanently available due to resource restrictions, and iii) the application subsystem comprising a single system bootstrapper (system seed). Each application usually has its own system seed that can be installed and un-installed separately. A system seed package typically includes:

1. a boot sensor
2. an optional boot actuator
3. an applications fixed set of guaranteed components

Usually, the application core system initialized by the system seed contains only a boot sensor specification and an administration component implementing that boot sensor. The administration component, usually a GUI, connects to the application origin server and from there downloads or updates a K-model XML file for the application and any necessary basic system components, that run in the domain of the application. These basic system components are necessary for providing a required minimal functionality of the system. This may include at least the necessary framework components, as well as a default context server, which handles the initial service communication by storing the messages to (persistent) context elements. Following the principles in this section, the minimal functionality can of course be extended on the fly, in case the corresponding resources for fulfilling additional services become available.

We refer to [MDF+04] for a conceptual introduction of the CAWAR framework, whereas a description of its technical realization can be found in [MFSS05].

3 Formal definitions

3.1 Foundations

Context adaptive systems are a subset of reactive systems. Hence, we do not introduce a new formalism for describing the system properties characterizing these systems. We rather demonstrate how related concepts like context adaptation can be formalized by means of an existing formalisms. This approach has two major benefits: on the one hand we can make use of a well understood formalism, which is furthermore already established. Consequently, interested readers do not have to acquire another formalism for understanding the described concepts.

The basis of our formalism is the FOCUS theory [BS01]. This theory treats systems as a composition of components $\mathbf{C}$ communicating via a set of channels $\mathbf{CH}$. Components access channels via ports. A port is denoted by a channel identifier $ch \in \mathbf{CH}$ and a reading operator (?) or a writing operator (!).

In a FOCUS specification all logical variables are typed. A type is nothing else than a set. Types are used to specify the set of messages M that can be sent along a channel ch or to constrain the set of possible internal states. They are also used to restrict the domains of variables. $TYPE$ thereby denotes the set of all types. Given some type $T \in TYPE$, $CAR(T)$ denotes the set of data elements of type T. $CAR(T)$ is called the *carrier set* for type T. We furthermore define a function $type : \mathbf{CH} \rightarrow T_{\mathbf{CH}}$ that returns the type of a channel.

Streams

For any set of messages M, we use M^{ω} to denote the set of all *streams* over M, in other words, the set of all sequences consisting only of messages from the set M. Moreover, M^{*} denotes the set of all finite streams over M, and M^{∞} denotes the set of all infinite streams over M. The set of infinite streams can be characterized by the set of functions F with $F : \mathbb{N}_{+} \rightarrow M$, whereby $\mathbb{N}_{+}$ denotes the natural numbers without 0. This means that the set of all streams is defined by $M^{\omega} = M^{*} \cup M^{\infty}$.

In order to model time-sensitive components, we need a way to express the timing of messages. For this purpose, we introduce the concept of timed streams. A timed stream differs from an ordinary (untimed) stream, in that it contains the history of messages transmitted within a certain time frame. A timed stream is represented by a set of function F_{time}, with $F_{time} : \mathbb{N}_{+} \rightarrow M^{*}$. $F_{time}(t)$ is the finite sequence of messages communicated within the time interval t. For any set of messages M, by $M^{\underline{\omega}}$, $M^{\underline{\infty}}$, and $M^{\underline{*}}$ we denote, respectively, the set of all timed streams over M, the set of all infinite timed streams over M, and the set of all finite timed streams over M. The set of all infinite timed streams is defined by $M^{\underline{\infty}} \stackrel{\text{def}}{=} (M^{*})^{\infty}$. Hence, $M^{\underline{\infty}}$ denotes all infinite sequence over finite sequences over M.

Channel assignments

For each set of typed channels, the channel assignment is the association of a channel and a stream, formally

$$\overrightarrow{\mathbf{CH}} \stackrel{\text{def}}{=} \{x \in (\mathbf{CH} \rightarrow M^{\underline{\infty}}) \mid \forall\, ch \in \mathbf{CH} : x(ch) \in CAR(type(ch))^{\underline{\infty}}\}$$

A channel assignment $x \in \overrightarrow{\mathbf{CH}}$ therefore assigns to each channel $ch \in \mathbf{CH}$ a timed stream of messages of $type(ch)$.

Components

With the definitions given above, we are now able to define components. Let $K \subseteq \mathbf{CH}$ be a given set of channels, then the set of components $C(K)$ using these channels K is defined by

$$C(K) \stackrel{\text{def}}{=} \{(I, O, F) \in (\mathcal{P}(K) \times \mathcal{P}(K) \times (\overrightarrow{\mathcal{P}(K)} \rightarrow \mathcal{P}(\overrightarrow{\mathcal{P}(K)}))) \mid F \in \overrightarrow{I} \rightarrow \mathcal{P}(\overrightarrow{O})\}$$

The behavior F of a component is therefore a relation between input- and output streams. I is the set of typed input streams, O the set of typed output streams. $F(x)$ is the set of all output histories for a $x \in \overrightarrow{I}$ that a component with behavior F is able to produce (nondeterminism is allowed).

The composition operator $\otimes$ for composing components is defined as

$$\forall\, C_1 = (I_1, O_1, F_1), C_2 = (I_2, O_2, F_2) \in C(\mathbf{CH}) : C_1 \otimes C_2 = (I, O, F)$$

so that

$I = (I_1 \cup I_2) \backslash (O_1 \cup O_2)$ and
$O = (O_1 \cup O_2) \backslash (I_1 \cup I_2)$ and
$F \in \overrightarrow{I} \rightarrow \mathcal{P}(\overrightarrow{O})$

with

$\forall\, x \in \overrightarrow{I} : F(x) = \{y' \in \overrightarrow{O} : \exists\, y \in \overrightarrow{I_1 \cup I_2 \cup O_1 \cup O_2} :$
$(y|_O = y') \wedge (y|_I = x) \wedge (y|_{O_1} = F_1(y|_{I_1})) \wedge (y|_{O_2} = F_2(y|_{I_2}))\}$

$x|_Q$ denotes the restriction of the assignment x to the channels in set Q.

3.2 Formalizing adaptation as feedback filter

The formalization of the adaptation basically splits into two parts:

- A definition of adaptation as a feedback filter
- A definition of the entities sensor, context, interpreter and actuator.

The basic formalization uses component networks that communicate via channels as a notion for systems or subsystems. The component behavior is specified by relations between input and output channels. Adaptation in this context can be explained as a change of such a network. Components or channels may be removed or added resulting in a changed system behavior.

As already mentioned, models are abstractions that are based on static assumptions. Thus, real dynamics can not be modeled. It can only be approximated by static assumptions. This is equal to the observation of unwanted behavior as a result of the frame problem (see [FSS06] for further details). Therefore dynamics is emulated by static models. Only observation from outside creates the illusion of dynamics.

By specifying a static model which switches between the visibility of different parts of the static overall behavior, dynamics can be created. Therefore, a system exposing an adaptive behavior formally must be regarded as a superposition of all its possible structures, functions and observable behaviors. The actual adaptation is then achieved by a filter, which exposes only certain parts of this superposition. Hence, these adaptations change the currently observable part of the overall system behavior.

As shown in figure 5 by using a filter all scenarios of changing the structure can be emulated.

3.3 Adaptation

Before starting with formalizing the filter and later context controlled filters, we introduce the helper function $\mathit{filter}_j : \{0, 1, \ldots, n\}^\omega \times MSG^\omega \rightarrow MSG^\omega$, which

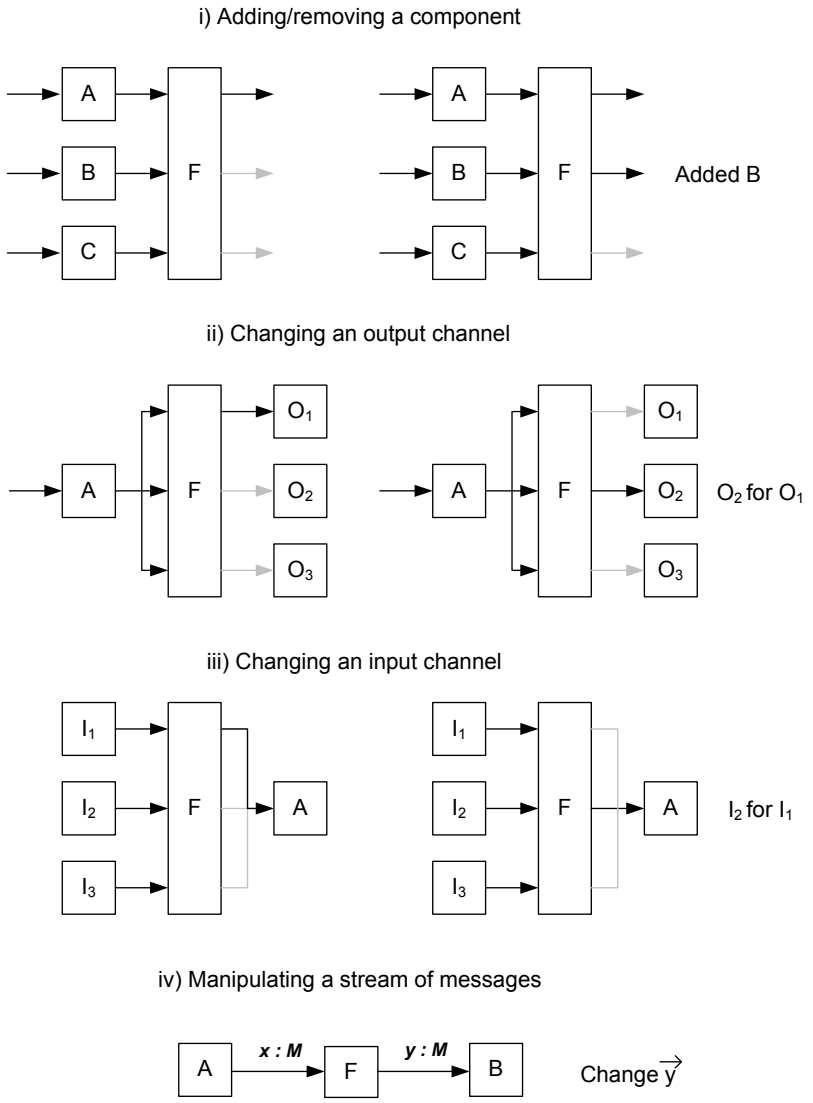


Fig. 5: Emulation of dynamics by filters

is defined as

$$a = j \Rightarrow \mathit{filter}_j(a\&p, b\&i) = b\&\mathit{filter}_j(p, i)$$
$$a \neq j \Rightarrow \mathit{filter}_j(a\&p, b\&i) = \mathit{filter}_j(p, i)$$

& thereby denotes a basic operator on streams, which is used in the infix notation and has the following signature:

$$\& : M \times M^\omega \to M^\omega$$

Given a stream s and a single message m, $m\&s$ describes the stream resulting from appending the message m to the head of the stream s. Another basic operator on streams is the *concatenation operator*. We use this binary operator

$$\frown : M^\omega \times M^\omega \to M^\omega$$

in infix notation as well. The concatenation of two streams produces a stream that starts with the messages of the first stream, followed by the messages of the second stream.

With this in mind, the function $filter_j$ reads a stream of messages $b\&i$ and gets a control sequence $a\&p$. If the first item a of the control sequence has the value of the filter-id j, then the first message b of the message stream is forwarded; otherwise it is ignored.

An output filter for a system as shown in figure 5 i) can be defined as the combination of filters F_{A0} for each output channel o of the system:

F_{A0}

in $i : MSG^{\omega}$

out $o : MSG^{\omega}$

$\exists\, p \in \{0,1\}^{\omega} : o \sqsupseteq filter_1(p, i)$

The equation within the body of specification F_{A0} denotes, that the filtered stream received on input channel i is a prefix of the stream send over output channel o. The oracle p represents a highly nondeterministic filter behavior. It generates a stream consisting of the control messages 0 and 1, which are produced in an arbitrary order. If the current message of p is a 1, then a message received on channel i is forwarded to channel o, otherwise the message is ignored. The following communication histories of the channels i, p and o illustrate the functioning of filter F_{A0}.

i : `abcdefghijklmnopqrst`
p : `01010101010101010101`
o : `bdfhjlnprt`

Multiple channels that should be filtered can be regarded to be multiplexed into one channel first. This is advantageous when we formalize calibration to avoid messing up different channels.

3.4 Context adaptation

Context adaptation denotes an adaptation, which is controlled by messages of the system environment known as context. Therefore, context adaptation is the extension of the general adaptation with the oracle p being replaced by external inputs.

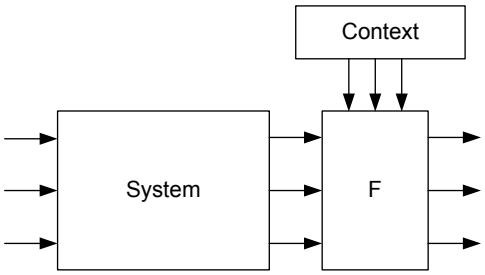


Fig. 6: Context adaptation as a context-controlled filter

A filter component F_{A0} is extended by a set of control channels $s \subseteq CH$. The control channels are fed by components that represent the environment. This results in the following filter F_{A1}.

F$_{\text{A1}}$

in $i : MSG^{\omega};\ s : 0,1^{\omega}$

out $o : MSG^{\omega}$

$\#o = \#(\{1\} \circledS (s|_{\#i}))$

$o \sqsupseteq \mathit{filter}_1(s|_{\#i}, i)$

The first equation states, that the number of messages send over channel o ($\#o$) complies with the number of messages with value "1" received on the control channel s with respect to the length of the input messages ($s|_{\#i}$). The context adaptation can now be constructed by an adequate combination of several filters.

3.5 Calibration

Calibration is the adaptation of the adaptation as described in Section 2.4. This can also be demonstrated on the formalism as proposed in figure 7.

- The adaptation behavior is determined by the information of the control channels.
- Calibration adapts the behavior that is observable at the control channels. Therefore the context is adapted.

This second adaptation is modeled as context adaptation as well. The information that feeds the control channels for the calibration is called calibration context.

This ensures that the first adaptation becomes adaptable. However the second one remains static and may be again subject to the frame problem. This is the case if the information on the control channels for the second adaptation is an interpretation made by the system rather being directly from an user.

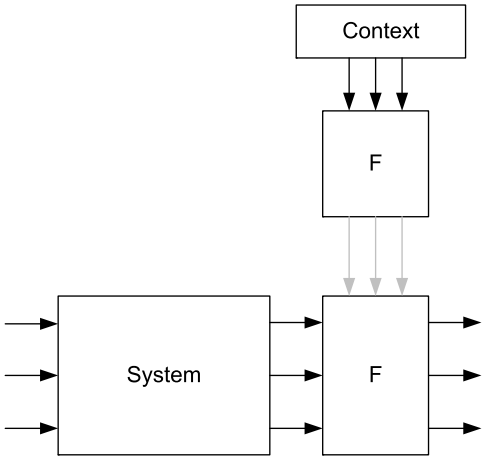


Fig. 7: Adaptation of adaptation

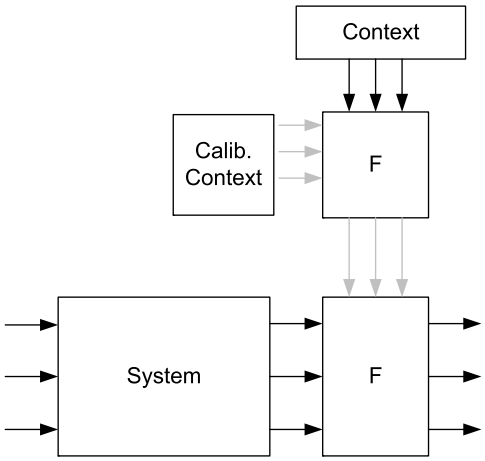


Fig. 8: Using calibration context for adapting the filter

Sometimes it is useful to automatize the calibration e.g., to construct multi user or learning systems. Due to the frame problem the automatic calibration requires a possibility of manual calibration as well. In a formal view, this could be done by concatenating calibrations. However this would render the formalization to be not generic. Therefore the calibration is constructed in a way to be able to be applied recursively. Thus the two context adaptations are combined as shown in figure 9.

Now the control information for the adaptation of the main system and the control information to select them originate from one context. Thus, context can be used to adapt parts of the context (from a system's point of view). To enable full adaptability it would be required that alternatives could be chosen for the control channels, which control the calibration. This is adequate to a change of different input channels. Of course, this is not possible with the filter

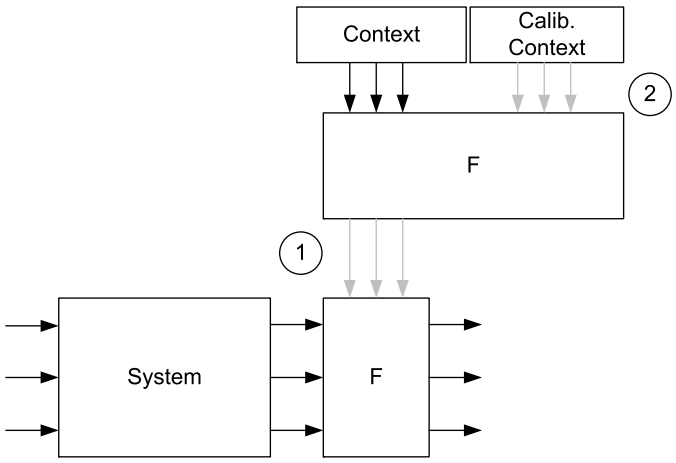


Fig. 9: Combination of context and calibration context

F_{A1}. We therefore introduce the filter $F_{A2(n)}$:

F A2(n)

in $i_1, \ldots, i_n : MSG^{\omega};\ s : 0, 1, \ldots, n^{\omega}$

out $o_1, \ldots, o_n : MSG^{\omega}$

$\forall\, k \in \{1, \ldots, n\} : \#o_k = \#(\{k\} \circledS (s|_{\#i_k}))$

$\forall\, k \in \{1, \ldots, n\} : o_k \sqsupseteq \mathit{filter}_k(s|_{\#i_k}, i_k)$

To enable a choice out of multiple input channels, a merge function is integrated into the filter component resulting in $F_{A3(n)}$:

F A3(n)

in $i_1, \ldots, i_n : MSG^{\omega};\ s : 0, 1, \ldots, n^{\omega}$

out $o : MSG^{\omega}$

$\forall\, k \in \{1, \ldots, n\} : \mathit{filter}_k(s|_{\#i_k}, i_k) \sqsupseteq \mathit{filter}_k(\{1, \ldots, n\} \circledS s, o)$

All outputs are multiplexed to one output channel. It is important to notice that even no output is possible ("0" on the control channel). The following example should clarify that.

Given $n = 2,\ i'_1 = \mathit{filter}(s, i_1),\ i'_2 = \mathit{filter}(s, i_2)$ *and* $s' = \{1, 2\} \circledS\, s$ then $F_{A3(n)}$ works as follows:

i_1 : `abcdefghijklmnopqrst...`
i'_1 : `behknqt...`
i_2 : `ABCDEFGHIJKLM...`
i'_2 : `CFIL...`
s : `012012012012012012012012...`
s' : `1212121212121212...`
o : `bCeFhIkL...`

The new filter $F_{A3(n)}$ is suitable to replace any other filters mentioned before in any scenario in figure 5. However, rather a set of channels with an identical syntactic interface is controlled than a single channel. But, F_{A3} either is able to calibrate itself by feedback or another context adaption. To enable both at the same time a combination of at least two F_{A3} is needed. Both need to be controlled via the same control channel. The control information must comprise the information for the controlling of the feedback loop as well as for the controlling of the other channel sets that are not fed back. The Filter $F_{A4(k,n_1,...,n_k)}$ therefore is:

$\mathrm{F}_{\mathrm{A4(k,n_1,...,n_k)}}$

in $\quad i_{11}, \ldots, i_{1n_1}, \ldots, i_{kn_k} : MSG^{\,\omega}$;
$\quad s : \{(s_1, \ldots, s_k) \mid s_1 \in \{0, 1, \ldots, n_1\}, \ldots, s_k \in \{0, 1, \ldots, n_k\}\}^{\,\omega}$

out $\quad o_1, \ldots, o_n : MSG^{\,\omega}$

$$\forall\, m \in \{1, \ldots, k\} : F_{A3}(i_{m_1}, \ldots, i_{mn_m}, \mathit{elemstrm}_m(s), o_m)$$

with

$$\mathit{elemstrm}_j : \{(s_1, .., s_k) \mid s_1 \in \{0, .., n_1\}, .., s_k \in \{0, .., n_k\}\}^{\,\omega} \to \{0, .., n_k\}^{\,\omega}$$

being a helper function with

$$0 < j \leq k \Rightarrow \mathit{elmstrm}_j((s_1, \ldots, s_k) \frown x) = s_j \frown \mathit{elemstrm}_j(x)$$

that extracts the control information for a set of channels $i_{k1}, \ldots, i_{kn_k}$ out of the complete control channel s and sends it to a F_{A3} filter for that set of channels controlling the associated output o_k.

An example for a filter F_{A4} may be the following:

For $k = 2$ (two sets of channels), $n_1 = 2,\ n_2 = 2$ and $s' = elemstrm_1\{s\}$ as well as o_1 for self calibration being fed back to s an example behavior is:

```
s   : (1,0) (1,1) (2,2) (2,1)...
s'  : 1122...
i11 : (1,0) (1,1) (2,0) (1,0)...
i12 : (2,0) (2,0) (2,2) (2,1)...
i21 : abcdefghijklmno...
i22 : ABCDEFGHI...
o1  : (1,0) (1,1) (2,2) (2,1)...
o2  : bCd...
```

Figure 10 shows the filter F_{A4} with the control of the calibration via a fed back channel (3). This channel defines which alternative channels for the control of the calibration of the adaptation is used (2). These channels control the adaptation of the system (1). The active channel of the calibration control is fed back to the calibration control channel. This allows the calibration control to hand over to an alternative channel.

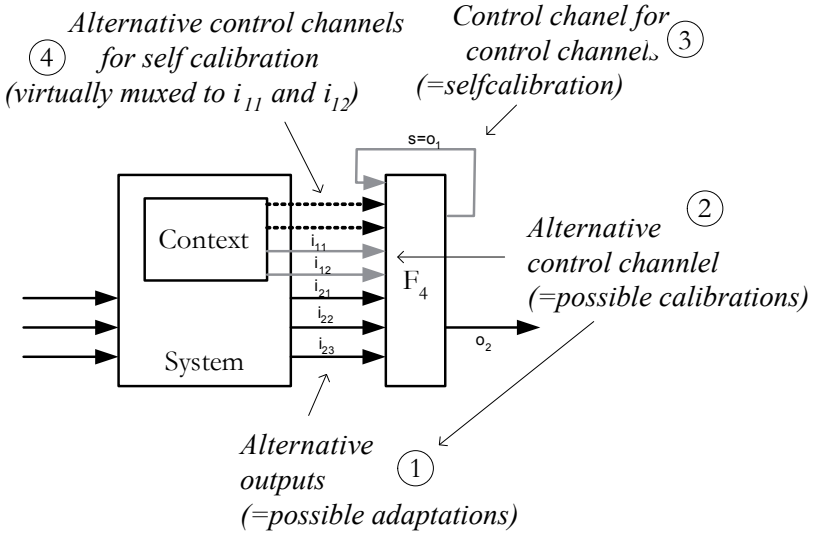


Fig. 10: Calibration with recursive feedback channel

3.6 Formalizing the framework infrastructure

In this section the calibrateable context adaptation will be formally defined. We will examine the following questions:

- How can potentially infinite alternative components be merged onto a

finite technical realization?

- How can the three process segments (obtaining context, making decisions and implementation of decisions) be mapped onto the formalization?
- How are the steps mapped onto sensors, interpreters, actuators and context elements?

Specification vs. implementation

The specification of the system must be complete and thus the complete behavior must be defined. But it is not necessary to claim all implementations of aspects of the system to be available at design time. If this was the case there would be a problem especially with the calibration if users add new behavior.

Instead, the model is defined with as few abstractions as possible. This model comprises all observable pieces of behavior and a mechanism to change between them. This is what adaptation actually is. Using a context, it is possible to have parts of the system being unimplemented at design time. Via the context adaptation it is possible to ensure that always adequate behavior is exhibited. This comprises that the behavior can be mapped on available resources (the availability of implementations is context as well).

To allow for runtime implementation, or at least to allow for binding implementations at runtime, there are some well known techniques in software engineering: dynamic binding of methods or dynamic loading of libraries. The most flexible technique are services (web services), which allows for exchanging component implementations.

Services and partial reconfigurability

Reconfiguration in software engineering is the (re-)arrangement of the communication structure of a predefined set of component implementations to archive a certain goal (to implement a function or satisfy a requirement). This is equivalent to changing the component implementation. (Web-)Services are an adequate technique to allow for component exchange. Services separate the component functionality from its implementation. Applications are only specified via their logical dependencies. At runtime, a suitable configuration with the set of implemented components is chosen and bound to the logical architecture. This is known as *Design@Runtime* [FSS00].

For the formal specification of services it is possible to use the basic model and to extend it with a structureless component (service) as proposed in [FSS00]. This allows to describe the behavior of a component according to the signature and to explicit bind it to a component implementation that realizes this behavior. A service is defined by using a partial behavior F_i' of a component in relation to the partial sets $I_i' \subseteq I, O_i' \subseteq O$ of the signature where the partial behavior must not interfere.

Definition

☞ A component $C = (I, O, F)$ **implements a service** if there is a service $D = (I_D, O_D, F_D)$ with $I_D \subseteq I,\ O_D \subseteq O$ such that for a function $F' : \overrightarrow{I'} \rightarrow \mathcal{P}(\overrightarrow{O'})$ with $I' = I \setminus I_D$ and $O' = O \setminus O_D$ holds:
Given $F_D : \overrightarrow{I_D} \rightarrow \mathcal{P}(\overrightarrow{O_D})$, then
$F' : \overrightarrow{I'} \rightarrow \mathcal{P}(\overrightarrow{O'}) \Rightarrow F = F_D \otimes F' \wedge I_D \cap I' = \varnothing \wedge O_D \cap O' = \varnothing$.

This implies that all services that are implemented by a component make use of disjoint channels. This idea allows the exchange of implementations but does not explain how this is done.

Definition

☞ An agent is an instance of a component $C = (I, O, F)$ and has a unique identifier $a \in ID$. The set of all agents over a set of channels $K \subseteq \mathbf{CH}$ is defined as:
$A(K) \stackrel{\text{def}}{=} \{(a, I, O, F) \in ID \times \mathcal{P}(K) \times \mathcal{P}(K) \times \overrightarrow{\mathcal{P}(K)} \rightarrow \mathcal{P}(\overrightarrow{\mathcal{P}(K)})) :$
$F \in \overrightarrow{I} \rightarrow \mathcal{P}(\overrightarrow{O})\}$
$a \in A(CH)$ with $a = (id, I_D, O_D, F_D)$ defines a single agent.

This definition helps in distinguishing between components and implementations. This is required to be able to distinguish between two instances of an identical implementation of a component. We can now draw the line to the formal definition of adaptation. An exchange of a technical component can be treated as a switch of alternative input or output channels. In fact this is two adaptations: an adaptation of the input channels for the service consumer and another adaptation of the output channels of the service provider.

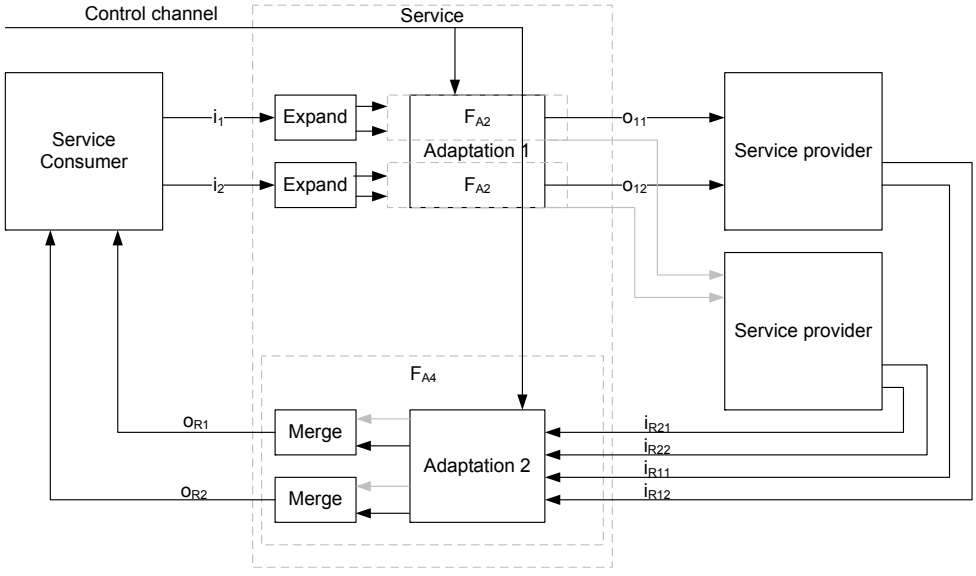


Fig. 11: Service as two adaptations

Figure 11 shows a service and the two mentioned adaptations. By reproducing the inputs, the input adaptation can be done by F_{A2} filters. The output channels can be handled by a F_{A4} filter. Only the control channel has to be prepared (corresponding information has to be extracted). With this considerations a service is defined by the specification $D(n, m, k)$.

D(n,m,k)

in	$i_1, \ldots, i_n : MSG^{\omega}$;	calling channels 1...n
	$i_{R11}, \ldots, i_{Rkm} : MSG^{\omega}$;	backward channels 1...m of 1...k components
	$s : \{0, 1, \ldots, k\}^{\omega}$	binding to 1...k service providers or 0
out	$o_{11}, \ldots, o_{km} : MSG^{\omega}$;	filtered output channels 1...n to 1...m
		service consuming components
	$o_{R1}, \ldots, o_{Rm} : MSG^{\omega}$	filtered and merged backward channels

$\forall\, j \in \{1, \ldots, n\} : F_{A2(k)}(\overbrace{i_j, \ldots, i_j}^{k}, s, o_{j1}, o_{jk})$	calling channels are duplicated and activated for one of 1...k targets
$\forall\, h \in \{1, \ldots, m\} : F_{A3(k)}(i_{R1h}, \ldots, i_{Rkh}, s, o_{Rh})$	back channels of 1...k service providers are activated

A service is a proxy component that redirects calls to one of k alternative service providing components and answers as if the caller component only communicates with one service providing component.

Definition

☞ The set of services D is defined as the set of components $C = (I_D \cup S, O_D, D(\mid I_D \mid, \mid O_D \mid, k)),\ k \in \{0..., \mid a \mid\}$
which allows for access to the set of agents $S = (id, I, O, F)$ of all components implementing a partial behavior $F_D\ C = (I, O, F)$ with $C = C_D \otimes C_1$ and $C_D = (I_D, O_D, F_D)$. Further defines $d \in D$ with $d = (I_D, O_D, F_D, k)$ a single service.

Definition

☞ The reconfiguration behavior of a service is defined by the stream of messages of the control channel. Therefore reconfiguration can be conceived as another component.

The logical architecture is an invariant over a reconfigurable system. Therefore, only partial reconfiguration is possible where the functional dependencies are not touched. This partial reconfigurability is suitable to suffice requirements about availability of functionality in adaptive systems which depend on extern resources and not on concrete implementations. This is true as long as a combination of services can be found that allows for implementing the defined (parts

of the) behavior.

Activator and total reconfiguration

Sometimes partial reconfiguration is not enough to ensure usefulness of the system. Therefore an super component is introduced that reads a description for behavior from the outside and represents it by means of internal behavior.

Definition

☞ A service is reconfigurable if the component that represents the service can be activated via a control channel

A service that is deactivated is not bound to a component and messages (except control messages) are dropped. The controlling of such reconfigurable services is like the controlling of the calibration. The activator service is defined by the following specification.

$\mathrm{AC}(\{\mathrm{d_1}(\mathrm{I_1}, \mathrm{O_1}, \mathrm{F_1}, \mathrm{n_1}), ..., \mathrm{d_k}(\mathrm{I_k}, \mathrm{O_k}, \mathrm{F_k}, \mathrm{n_k})\} \in \mathcal{P}(\mathrm{D}))$

in $s_{all} : \{(s_1, \ldots, s_k) \mid s_1 \in \{0, 1, \ldots, n_1\}, \ldots, s_k \in \{0, 1, \ldots, n_k\}\}^{\omega}$

out $s_{D0} : \{0, 1, \ldots, n_1\}^{\omega}, \ldots, s_{Dk}\{0, 1, \ldots, n_k\}^{\omega}$

$\forall\, m \in \{0, \ldots, k\} : s_{Dm} = elemstrm(s_{all})$

Definition

☞ The activator constructs control information from a control channel for all services including the activator service, which is implemented by itself.

If all dependencies of the components of a given set of agents are specified in terms of services, the activator could emulate as well the change of the logical architecture as the change of component implementations. The activator can be modeled as a component – and thus can be denoted as context (cf. section 3.5).

3.7 Formalizing the context adaptation model

The modeling of context adaptive systems by means of the formalization is quite complex and does not reflect appropriately the three steps in adaptation. To solve this issue, we will have to make some further investigations.

The activator only has to be able to adapt services that are part of the adaptation logic, and not of the core system (cf. section 2.7). This implies that only services are affected that participate in obtaining, processing and using context. These are exactly the sensors, interpreters, actuators and context elements from the K-Model in section 2.5.

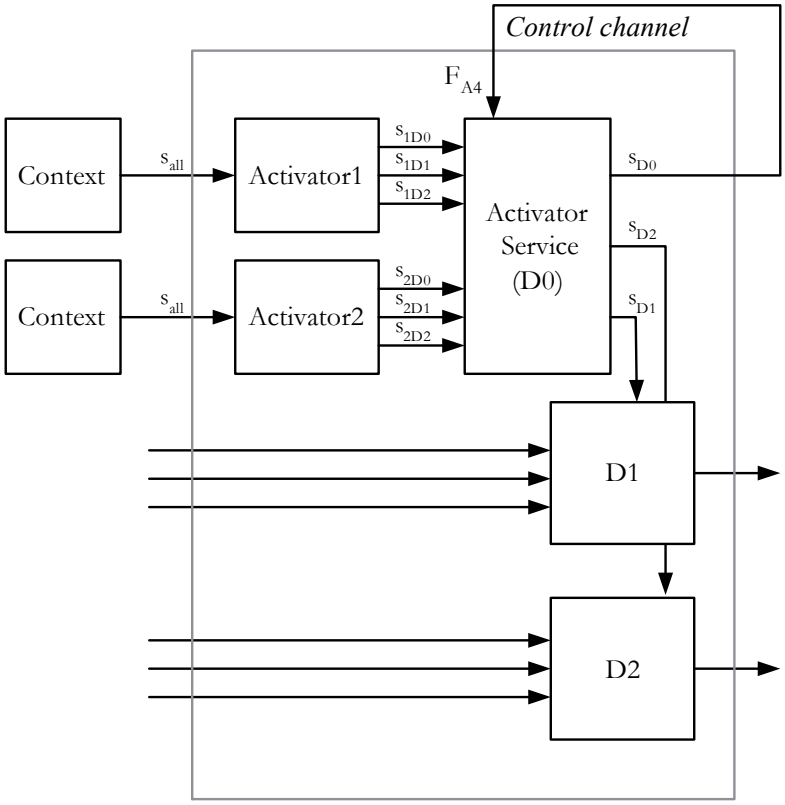


Fig. 12: Calibrateable adaptation with activator

The K-Model was conceived to require decoupling of components. They should only be able to communicate via context. This results in a unified interface. Therefore the calibrateable context adaptation may be constructed by these four service types and independently of the application.

Context elements

Context elements can be formalized by the following specification, whereby for all subsequent specifications $CTX \subset MSG$ and $T \subset \mathbb{N}$.

$\mathrm{F}_{\mathrm{ctx}}$

in $\quad i : (CTX \times T)^{\omega}, r : T^{\omega}$

out $\quad o : CTX^{\omega}$

$\forall\, k, j \in \{0, \ldots, \#i\};\ k < j;\ (c_1, t_1), (c_2, T_2) \in CTX \times T :$

$(c_1, t_1) = pos(i, k) \wedge (c_2, t_2) = pos(i, j) \Rightarrow t_1 < t_2$

$o \sqsupseteq gen(i, r)$

The function $gen : (CTX \times T)^{\omega} \times T^{\omega} \rightarrow CTX^{\omega}$ is a helper function defined by

$$gen(i, a\&r) = choose(expand(i, 0, \perp), a)\&gen(i, r)$$

which selects the context from the input stream i according to a certain time stamp and arranges i to an output stream, that corresponds to an input stream r of time stamps.

The helping function $choose : (CTX \times T)^{\omega} \times T \rightarrow CTX$ with

$$t_1 = t \Rightarrow choose((c_1, t_1)\&e, t) = c_1$$
$$t_1 > t \Rightarrow choose((c_1, t_1)\&e, t) = choose(e, t)$$
$$t_1 < t \Rightarrow choose((c_1, t_1)\&e, t) = \perp$$

selects the value of the context at a certain time stamp.

Another function $expand : (CTX \times T)^{\omega} \times T \times CTX \rightarrow (CTX \times T)^{\omega}$ with

$$t_1 = t \Rightarrow expand((c_1, t_1)\&i, t, c) = (c_1, t_1)\&expand(i, t + 1, c_1)$$
$$t_1 > t \Rightarrow expand((c_1, t_1)\&i, t, c) = (c, t)\&expand((c_1, t_1)\&i, t + 1, c)$$

extends a stream of context information and timestamps by intermediate results. Context information is repeated until another context information with higher time stamp is available (e.g., (a,1),(b,3) becomes (a,1),(a,2),(b,3)).

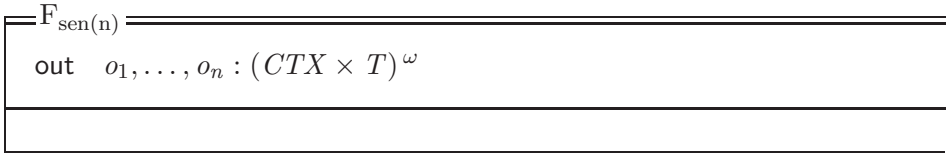
After all, the helper function $pos : CTX^{\omega} \times T \rightarrow CTX$ defined by

$$p > 1 \Rightarrow pos(a\&i, p) = pos(i, p - 1)$$
$$p = 1 \Rightarrow pos(a\&i, p) = a$$

returns the element at the specified position within a stream.

Sensors

Sensors are defined by the following specification $F_{sen(n)}$:



According to this specification, sensors expose an arbitrary partial component behavior, that may generate context information at their corresponding output channels.

Actuators

Actuators in turn are defined by the specification $F_{act(n)}$.

$\mathrm{F}_{\mathrm{act(n)}}$

in	$i_1, \ldots, i_n : CTX^{\,\omega}$
out	$r_1, \ldots, r_n : T$
	$\forall\, k \in \{1, \ldots, n\} : \#i_k = \#r_k$

Actuators are arbitrary components, which can request context information from a context element. They need to be able to generate time stamp information.

Interpreters

Interpreters are defined by the specification $F_{int(n,m)}$

$\mathrm{F}_{\mathrm{int(n,m)}}$

in	$i_1, \ldots, i_n : CTX^{\,\omega}$
out	$r_1, \ldots, r_n : T, o_1, \ldots, o_m : (CTX \times T)^{\,\omega}$
	$\forall\, k \in \{1, \ldots, n\} : \#i_k = \#r_k$

Interpreters integrate the functionality of sensors and actuators. They are used to interpret context information and to generate new context information from the given one.

4 Conclusion

In this document we have described a formal model for the notion of calibrateable context adaptation and provided an overview of important issues and how they could be solved. The objective was to provide a formal framework for constructing context aware systems. We informally introduced and motivated several notions related to context awareness. With the informal notions in mind, we described the so called K-Model, which allows to structurally describe the adaptation logic of context aware systems. We proposed to separate those systems into two parts in order to reduce the overall complexity and to explicitly present relevant parts of their decision making to users. One part comprises the adaptation logic and the other part constitutes the system core.

The intuitive description was used to formalize adaptation as a filter. Metaphorically speaking, this filter acts like a window or magnifying glass. The filter

"scrolls" over the overall behavior of the system, thereby only exposing a certain portion at a particular time. This technique is used for simulating dynamic behavior. Depending on the situation of use, the window selects the appropriate part of the overall system behavior by changing its position. In consequence, another part of the overall system behavior becomes observable. We formalized this approach by means of the FOCUS theory for specifying interactive systems.

The K-Model provides a structural view on adaptation and describes the information flow, which is responsible for the decision making in order to adapt to changing environment conditions. To emphasize this, the K-Model does not provide the exact information for determining how decisions are made. Further work directs to issues concerning how situations are assessed, which conclusions are drawn and, finally, how decisions are made. The K-Model as a framework allows to incorporate this information in a syntactic form (cf. syType in section 2.7) by using it as part of the behavioral specifications of an entity. On the other hand – since the CAWAR framework is intended to support the implementation of context aware applications – it is more convenient to specify the overall behavior of the system in a top-down approach. This overall specification can afterwards be mapped to an appropriate K-Model describing the structure and communication between the involved entities of the adaptation logics.

References

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[Den84] Daniel C. Dennett. Cognitive wheels: The frame problem of ai. *Minds, Machines, and Evolution*, pages 128–151, 1984. Cambridge University Press.

[FLSS08] Michael Fahrmair, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. Adaptation design in ubiquitous computing. techreport, 2008.

[FSS00] Michael Fahrmair, Chris Salzmann, and Maurice Schoenmakers. Carp@ - a reflection based tool for observing jini services. In *Proceedings of the 1st OOPSLA Workshop on Reflection and Software Engineering*, pages 209–227, London, UK, 2000. Springer-Verlag.

[FSS05] M. Fahrmair, W. Sitou, and B. Spanfelner. An engineering approach to adaptation and calibration. In *Modeling and Retrieval of Context MRC 2005*. Springer LNCS 3946, 2005.

[FSS06] M. Fahrmair, W. Sitou, and B. Spanfelner. Unwanted behavior and its impact on adaptive systems in ubiquitous computing. In *14th*

Workshop on Adaptivity and User Modeling in Interactive Systems – LWA/ABIS, 2006.

[MDF+04] E. Mohyeldin, M. Dillinger, M. Fahrmair, P. Dornbusch, and W. Sitou. Interworking between link layer and application layer adaptations in a reconfigurable wireless middleware. In *15th IEEE International Symposium on Personal, Indoor and Mobile Communications (PIMRC 2004, Barcelona/Spain)*, sep 2004.

[MFSS05] Eiman Mohyeldin, Michael Fahrmair, Wassiou Sitou, and Bernd Spanfelner. A generic framework for context aware and adaptation behaviour of reconfigurable systems. In *The 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC05), 11-14 September 2005, Berlin, Germany*, 2005.

[SBHW99] Albrecht Schmidt, Michael Beigl, and H. Hans-W. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999.

[Sch02] Albrecht Schmidt. *Ubiquitous Computing - Computing in Context*. PhD thesis, Computing Department, Lancaster University, England, U.K., 11 2002.

[SFS06] Alistair Sutcliffe, Stephen Fickas, and MacKay Moore Sohlberg. PC-RE: a method for personal and contextual requirements engineering with some experience. *Requirements Engineering*, 11(No. 4):157–173, 2006.

[Sim69] Herbert A. Simon. *The sciences of the artificial*. MIT Press, Cambridge, MA, USA, 1969.

[SS07] W. Sitou and B. Spanfelner. Towards requirements engineering for context adaptive systems. In *The thirty-first Annual International Computer Software & Application Conference (COMPSAC). Volume 2 - Workshop Papers*, 2007.

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.