Kalibrierbare Kontextadaption für Ubiquitous Computing

Michael Robert Fahrmair

Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching
Germany

Institut für Informatik
der Technischen Universität München

Kalibrierbare Kontextadaption für Ubiquitous Computing

Michael Robert Fahrmair

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Uwe Baumgarten

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2. Univ.-Prof. Gudrun J. Klinker, Ph.D.

Die Dissertation wurde am 3.11.2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.01.2005 angenommen.

Kurzfassung

Ubiquitous Computing[1] wird oft gemeinhin als vierte Welle der Computernutzung nach den Mainframes, PCs, und Mobilgeräten bezeichnet. Kennzeichen sind eine kontinuierliche Nutz- und Verfügbarkeit der Anwendung in unterschiedlichsten Situationen von Hardwareverfügbarkeit und bewusster oder unbewusster Nutzerinteraktion. Neue Softwaretechnologien und Interaktionsformen mit teilweise höherem Automatisierungsgrad können die daraus resultierenden Probleme aber nur in Einzelsituationen lösen, tragen jedoch nichts zur situationsübergreifenden Ubiquität, also den Wechsel zwischen heterogenen Anwendungssituationen bei. Kontextadaption ist dagegen ein generisches technisches Verfahren, Funktionalität, Benutzerschnittstelle und Realisierung eines Softwaresystems abhängig von Informationen über dessen Anwendungssituation zu verändern. Dies schließt auch die situationsbedingte Auswahl von semi- oder vollautomatischen (autonomen) Interaktions- und Implementierungsformen mit ein. Kontextadaption kann man daher auch als bedarfs- und situationsgerechte Automatisierung und damit als Grundlage der Ubiquität verstehen.

In der Praxis zeigt sich jedoch, dass kontextadaptive Systeme im realen Einsatz trotz ihrer Anpassungsfähigkeit aus Sicht der Benutzer ein überproportional häufiges Auftreten von unerwünschtem oder unerwartetem Verhalten zeigen. Bedingt ist dies durch perspektivische oder zeitliche Divergenzen in den Wahrnehmungsmodellen der Entwickler und Benutzer. Diese Divergenzen nehmen mit steigender Adaptivität eines Systems zu. Gleichzeitig reduzieren steigende Automatisation und eine benutzerzentrierte Ausrichtung die sonst vorhandenen impliziten Kompensationsmechanismen. Der Benutzer passt sich folglich nicht mehr an die Tücken der Software an, sondern es wird erwartet, dass sich die Software an die Bedürfnisse des Benutzers anpasst.

Die vorliegende Arbeit zeigt die technischen Zusammenhänge dieser Problematik und wie diese durch einen expliziten Kalibrierungsmechanismus umgangen werden kann. Grundlage dafür ist eine höhere Transparenz des Adaptionsverhaltens für den Benutzer und die Möglichkeit, Adaption selbst für eigene oder geänderte Bedürfnisse zu modifizieren. Realisiert wird die Kalibrierung auf Basis eines formal fundierten Modells, das zusammen mit geeigneten Beschreibungs- und Konstruktionstechniken in ein Framework eingebettet ist. Das Framework erlaubt wiederum den einfachen Entwurf von ubiquitären Anwendungen, deren Adaptionsverhalten auch zur Laufzeit an spezifische Bedürfnisse und Besonderheiten kalibriert werden kann.

[1] und synonyme Begriffe wie Pervasive/Invisible/Context Aware/Human Centric Computing/Ambient Intelligence

Danksagung

Zuerst möchte ich mich bei Prof. Manfred Broy für das Ermöglichen der Arbeit und die begleitende Betreuung bedanken. Dank gilt ebenso Prof. Gudrun Klinker für die Übernahme der Zweitbegutachtung.

Bei meinen zeitweisen und dauerhaften Kollegen aus dem Kompetenzzentrum für Mobility & Context Awareness Christian Salzmann, Maurice Schoenmakers, Peter Dornbusch, Norbert Diernhofer, Wassiou Sitou und Sebastian Winter möchte ich mich für viele Diskussionen und Anregungen bedanken.

Mein Dank geht ebenso an unsere langjährigen Projektmitarbeiter Eiman Mohyeldin und Markus Dillinger von Siemens ICM N für die gute Zusammenarbeit und ihre Geduld, die sie besonders in der Endphase dieser Arbeit bewiesen haben.

Bei dem dem gesamten Lehrstuhl möchte ich mich für die gute Atmosphäre und die unkomplizierte Zusammenarbeit bedanken.

Schließlich und endlich möchte ich auch noch meiner Frau dafür danken, dass sie all die Jahre geduldig auf die Fertigstellung dieser Arbeit gewartet und auf dem Weg dahin das Leben in einem manchmal nicht ganz so "intelligenten Haus" klaglos ertragen hat.

Zur Benutzung dieser Arbeit

Aus Gründen der besseren Übersichtlichkeit wurden über die Arbeit hinweg verteilt die folgenden Strukturierungs- und Auszeichnungsmerkmale verwendet:

Kernfragen

In der Kapitelunterschrift jedes Kapitelanfangs werden Kernfragen aufgezählt, die dann im jeweils nachfolgenden Text behandelt werden (Abb. 1).

Abb. 1: Kernfragen als Kapitelunterschriften

Dies erleichtert es, etwaigen spezifischen aufgetretenen spezifischen Fragestellungen direkt in den einzelnen Kapiteln nachzugehen.

Randmarken

Zum Zwecke der Strukturierung längerer Textpassagen wurden die einzelnen Punkte einer Argumentationskette durch Randbemerkungen markiert (Abb. 2).

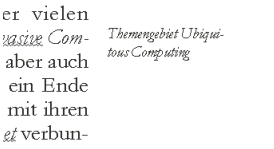


Abb. 2: Randbemerkungen

Auf diese Weise können dem individuellen Leser bereits bekannte Erklärungen oder Grundlagen leicht übersprungen werden. Zusätzlich weisen solche Randmarken auf Zwischenergebnisse hin, die in späteren Kapiteln wieder aufgegriffen werden.

Definitionen

Definitionen beschreiben jeweils die Essenz eines für diese Arbeit gültigen Verständnisses wichtiger Begriffe (Abb. 3).

Definition 1: *Ubiquitous Computing*

Ubiquitous Computing ist die direkte oder indirekte Nutzung computerbasierter Anwendungssysteme in möglichst vielen Situationen (z.B. Orte, Zeiten, Tätigkeiten) eines Benutzers.

Abb. 3: Definitionen

Für eine bessere Verweisbarkeit sind Definitionen mit einem eigenen Nummerierungsschema ausgezeichnet.

Beispiele

Beispiele illustrieren einen zuvor beschriebenen, meist allgemein gültigen oder abstrakten Zusammenhang anhand eines konkreten Anwendungsfalls. Dieser entstammt in der Regel aus der in Kapitel 2 beschriebenen Fallstudie einer ubiquitären Smarthome Anwendung (Abb. 4).

se dies in einer bestimmten Situation gerade erfordern, nicht mehr auf das gesamte System.

Die automatische Übertragung und damit Adaption eines gerade noch mobil auf einem Smartphone betrachteten Informationsdienstes auf einen nach Betreten eines Raumes verfügbaren stationären Bildschirm findet im Hintergrund statt. Die Nutzung des Dienstes kann aber durchaus die direkte und bewusste Aufmerksamkeit des Benutzers erfordern. Trotzdem kann durch die Adaption die Nutzbarkeit erhöht, oder aufgrund der größeren Betrachtungsfläche überhaupt erst hergestellt werden.

Abb. 4: Beispiele aus der Fallstudie

Um diese Durchgängigkeit der Beispiele zu betonen, sind solche Bezüge auf die Fallstudie farblich vom Rest des Textes abgehoben.

Quelltexte

Zeilenorientierte formalsprachliche Quellen sind mit einer separaten Zeilennummerierung versehen (Abb. 5).

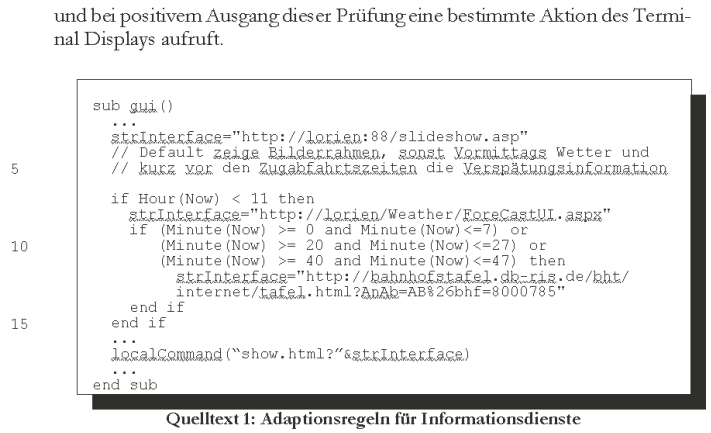


Abb. 5: Nummerierte Quelltexte

Dies ermöglicht ein späteres Zitieren oder Verweise auf einzelne Ausschnitte des jeweiligen Quelltextes.

Inhaltsverzeichnis

Teil I

Einführung, Begriffsbildung und Problematik

KAPITEL 1 Einleitung und Begriffe

Was ist das Themenfeld und die Motivation dieser Arbeit? Welcher Problematik wurde nachgegangen und welche Ergebnisse wurden erzielt?

Themengebiet Ubiquitous Computing

Ubiquitäre Computeranwendungen sind gegenwärtig unter den verschiedensten synonymen Bezeichnungen (*Ubiquitous-*, *Autonomous-*, *Ambient* oder *Pervasive Computing*) Gegenstand der wissenschaftlichen oder industriellen Forschung aber auch besonders des öffentlichen Medieninteresses. Nicht selten wird sogar schon so weit gegangen, ein Ende der Ära der Personalcomputer (PCs) zu propagieren. Anstelle der PCs mit ihren isolierten Plattformen und Anwendungen scheinen die zum *Evernet* verbundenen "allgegenwärtigen Computer" und die damit einhergehende, selbstständig, intelligent und anpassungsfähig handelnde Software den nächsten logischen Schritt in der Evolution der Informationstechnologie (IT) zu markieren.

1.1 Ubiquitous Computing

Eine solche mögliche Entwicklung ist zum momentanen Zeitpunkt aber höchstens ansatzweise absehbar. Bis dato werden nämlich in der Fachliteratur genauso wie auch in den populärwissenschaftlichen Publikationen noch immer sehr unterschiedliche Konzepte und Vorstellungen mit dem Begriff der Ubiquität in Verbindung gebracht. Eine einheitliche Umsetzung ist unter solchen Umständen natürlich erst recht in weiter Ferne zu sehen.

Neben einem manchmal übertriebenen Wunsch nach Alleinstellungs- und Abgrenzungsmerkmalen liegen diese terminologischen Differenzen vor allem auch darin begründet, dass bereits die reine sprachliche Bedeutung des Begriffes in Verbindung mit Computern lediglich eine nicht näher spezifizier-

te Überall-Nutzung von Computern suggeriert (z.B. [2], [3],[4]). Dieses Begriffsverständnis ändert sich auch nicht durch das Hinzuziehen fachspezifischer Interpretationen aus anderen Gebieten als der IT (z.B. [5]), sodass durch das intuitive Begriffsverständnis nicht eindeutig festgelegt ist, ob es sich bei *Ubiquitous Computing* um jeweils einen Computer handelt, der "überall" (örtlich) beziehungsweise im Sinne von "für alles" (jede Tätigkeit) genutzt werden kann, oder aber lediglich um eine große Zahl von überall vorhandenen Computern. Ebenso wenig leitet sich aus dem intuitiven Begriffsverständnis ohne Hinzuziehen eines technischen Zusammenhangs her, ob der Begriff aus der Sicht nur eines oder sehr vieler Benutzer formuliert ist.

Bisher keine einheitliche Definition von Ubiq. Computing

Diese intuitive Verständnisunsicherheit überträgt sich naturgemäß auch auf die Definition und Verwendung des Begriffes Ubiquitous Computing im Fachgebiet der IT. Da sich bisher keine Definition fachlich eindeutig etablieren konnte, steht der Begriff dort auch weiterhin für sehr unterschiedliche Konzepte der Informationsverarbeitung. So wird Ubiquitous Computing einmal als Interaktionen mit in die Umwelt verteilten Computern [37] bezeichnet, ein anderes Mal als die Konvergenz von PDAs und Mobiltelefonen [38]. Genauso kann der Begriff auch als ein, von überall her möglicher Zugriff auf Informationen unter Benutzung verschiedener Geräte verstanden werden. Manchmal bezeichnet Ubiquitous Computing dagegen die Schaffung einer totalen *Konnektivität* [39] durch Vernetzung aller natürlichen und künstlichen Gegenstände und Personen. Oft wird der Begriff aber auch nur in Ermangelung einer klaren Definition als eine unspezifische Post-PC beziehungsweise Post-PDA Ära oder so genannte Dritte (Vierte) Welle der Computer Nutzung nach Mainframes, PCs und einzelnen tragbaren Geräten beschrieben. Diese neue Welle ist dann meist durch die Tatsache charakterisiert, dass eine Person im Gegensatz zur Ära der Mainframes und PCs mehrere Computer nutzt. Ob diese Nutzung gleichzeitig sein muss, oder einfach nur bedeutet, dass eine Person mehrere stark spezialisierte Computer bis hin zu intelligenten Kühlschrankmagneten [6] besitzt und jeweils das für einen jeweiligen Zweck gerade am besten geeignetste Gerät auswählt, bleibt dabei offen. Ebenso ob diese Geräte ständig mit sich herumgetragen werden (vernetzte Kleidung [7]), oder Teil der Umgebung sind, ob sie explizit genutzt werden können, oder implizit über eine einheitliche Schnittstelle [33].

Viele Synonyme

Ein ähnliches Bild zeichnet sich bei verwandten oder alternativen Begriffen. Entweder basieren deren Abgrenzungen auf wiederum neuen und sehr eng gefassten Definitionen von Ubiquitous Computing oder die Begriffsdefinition ist identisch mit bereits existierenden Definitionen von Ubiquitous Computing, beziehungsweise wird sogar direkt mit diesen gleichgesetzt (z.B. *Pervasive Computing* [39])

Analysiert man die verschiedenen existierenden Definitionen von Ubiquitous Computing und die synonymer Begriffe, zeigt sich jedoch eine Gemeinsamkeit. Sie sind in der Regel Ausdruck für verschiedenste Ausprägungen einer gesteigerten Nutzung von Computern und Computeranwendungen. Dies kann sich zeitlich und örtlich als gesteigerte Nutzung im Vergleich zu einem PC am Arbeitsplatz oder zu Hause äußern. Ebenso aber auch als gesteigerte mobile Nutzung unterwegs. Diese Grundidee ist aber auch gleichzeitig die Ursache der unterschiedlichen Definitionen. Sowohl der Gegenstand der Definition, nämlich die Anwendung, als auch der definierte Vorgang der Mehrbenutzung besitzen mehr als eine Bedeutungsdimension. So besteht eine Anwendung meistens aus Hardware und Software (siehe Def. 3). Mehrnutzung kann daher durch die erhöhte Nutzung einer einzelnen, oder aber die gleich bleibende Nutzung einer größeren Zahl von gleichzeitigen Anwendungen erreicht werden. In Anhang A1 wird diese Topologie der mehrdimensionalen Bedeutungsräume von Ubiquität zum Zwecke der besseren Einordnung der Ergebnisse dieser Arbeit noch einmal genauer betrachtet.

Für die Einführung der im Folgenden zu motivierenden Problematik ist jedoch eine einfachere Definition vonnöten, welche die verschiedenen Aspekte existierender Definitionen in sich vereint und auf das Wesentliche fokussiert. Dies gelingt nur, indem der Begriff auf die hinter den unterschiedlichen existierenden Definitionen verborgene gemeinsame Motivation reduziert wird, ohne sich dabei auf einen von mehreren gleichberechtigten Wegen festzulegen, dieses Ziel zu erreichen:

Definition 1: *Ubiquitous Computing*

Ubiquitous Computing ist die direkte oder indirekte Nutzung computerbasierter Anwendungssysteme in möglichst vielen Situationen (z.B. Orte, Zeiten, Tätigkeiten) eines Benutzers.

Situative Definition von Ubiq. Computing

Um eine Unterscheidung zwischen dieser allgemeinen zielorientierten und den vielen anderen lösungsmotivierten Definitionen zu ermöglichen, wird diese Art der Festlegung im Folgenden als *situative Definition* von Ubiquitous Computing bezeichnet. Die darin enthaltenen Begriffe *System* und *Anwendung* werden dabei im weiteren Verlauf der Arbeit so verstanden, dass ein ubiquitäres System aus einer ubiquitären Anwendung und einer technischen Hardware-Umgebung besteht, die nicht notwendigerweise eine Einheit bilden:

Definition 2: *System*

Ein System ist ein abstrakter und aus einer bestimmten Perspektive explizit von seiner Umgebung abgegrenzter Gegenstand [15]. Systeme sind aus Teilen (Systemkomponenten oder Subsystemen) zusammengesetzt, die untereinander in verschiedenen Beziehungen stehen können. Nicht weiter zerlegbare Systemteile werden als Systemelemente bezeichnet [14].

Definition 3: *Anwendung*

Eine Anwendung ist ein System, das Software-Komponenten enthält, die zur Lösung einer Menge von inhaltlich zusammengehörigen Aufgaben notwendig sind [14][15].

Ubiquitäre Anwendungen sind dynamisch bezüglich ihrer HW und SW

Die Idee, dass eine ubiquitäre Anwendung mit wechselnden Hardwaresystemen interagiert, lässt sich darüber hinaus auch noch auf die Software selbst anwenden. So sind ubiquitäre Systeme denkbar, die aus mehreren auch wechselnden ubiquitären (Teil-)Anwendungen zusammengesetzt sind. Man spricht dann generell von einem dynamischen System, dessen Zusammensetzung aus Hardware- und Softwarekomponenten veränderlich ist. Damit wird klar, dass der Situationsbezug aus Definition 1 zwar aus der Perspektive des Benutzers formuliert wird, nicht jedoch auf den Benutzer beschränkt bleibt, sondern auch das System selbst mit einbeziehen kann. Die Nutzbarkeit (erste Ubiquitätsbedingung) in möglichst vielen Situationen aus Definition 1 ist damit durch die Herstellbarkeit einer technischen Verfügbarkeit als notwendige Bedingung (zweite Ubiquitätsbedingung) theoretisch begrenzt. Die Nutzbarkeit definiert sich damit durch eine, bezogen auf die Benutzerbedürfnisse, richtige Auswahl für den Fall, dass mehrere technische Realisierungsmöglichkeiten zur Auswahl stehen.

1.1.1 Vergleich der situativen Definition mit dem historischen Ursprung des Begriffes Ubiquitous Computing

Ursprüngliche Definition von Marc Weiser als unsichtbare Nutzung

Der Ursprung der momentan sehr zahlreich im Umlauf befindlichen unterschiedlichen Definitionen und Deutungsweisen von Ubiquitous Computing war überraschender Weise eine einzige sehr klar abgegrenzte Begriffsdefinition von Marc Weiser. Dieser untersuchte eine mögliche neue Generation von Computernutzung [8]. Sein Ziel dabei war, durch die für den Benutzer unsichtbare Verfügbarkeit vieler Computer in der physischen Umwelt die Nutzung von Computern zu verbessern [34]. Die Computer selbst sollten sich dabei unauffällig in die Lebensgewohnheiten der Benutzer integrieren und sich simultan während anderer Tätigkeiten benutzen lassen. Weisers Hoffnung war, dass dies durch den Benutzer dann lediglich unterbewusst im weitesten

Sinne als Verbesserung der eigenen Fähigkeiten empfunden werden würde, ähnlich wie das bei fortschrittlichen Bremsassistenten oder anderen elektronischen Fahr- und Sicherheitshilfen, wie beispielsweise automatischen Unfallrettungssystemen [36] im Automobilbereich der Fall ist.

Die Forderung nach weitgehender unbewusster Nutzbarkeit, vergleichbar dem Geräusch eines Motors, das nur unterbewusst wahrgenommen wird, bis es auf ein mögliches technisches Problem hindeutet [35], diente Weiser auch für eine klare Abgrenzung seines Ubiquitous Computing von anderen Strömungen der Post-PC Ära wie Personal Digital Assistants (PDA) und Agententechnologien im Speziellen oder *Mobile Computing* ganz im Allgemeinen [34]. Leider waren die Abgrenzungsversuche in der Praxis ungeeignet. Weiser selbst musste einräumen, dass die meisten prototypischen Versuche, Computernutzung vollständig in die periphere Hintergrundwahrnehmung zu verlagern bis auf einige wenige Ausnahmen (z.B. der „Dangling String“ [36]) erfolglos waren [8]. Diese Schwierigkeiten waren im Übrigen bereits ein erster Hinweis auf eine Problematik, die im Zentrum des Interesses der vorliegenden Arbeit liegt. Es wird sich zeigen, dass das Problem inhärent mit der über die situative Definition ausgedrückten Grundidee des Ubiquitous Computing verknüpft ist, egal, in welcher technischen Definition diese ihren Ausdruck findet. Unterschiede zeigen sich lediglich in der Intensität, in der das Problem wahrgenommen werden kann. Ganz offensichtlich war Marc Weisers ursprüngliche Sichtweise aber sehr stark von dieser Problematik betroffen.

Erste Hinweise auf Probleme bei der praktischen Umsetzung.

Aufgrund dieser frühen Schwierigkeiten, eine sinnvolle, weil funktionierende Begriffsabgrenzung zu finden, setzte sich der Begriff des Ubiquitous Computing dann mehr und mehr als Sammelbegriff für verschiedene Abschwächungen der ursprünglichen Idee von Weiser durch. Das geht bis hin zu einem Punkt, an dem das Ubiquitous Computing nur noch sehr hardwarebezogen als lediglich infrastruktureller Rahmen definiert wird, innerhalb dessen verschiedene Interaktionstechnologien existieren können. Die Infrastruktur selbst wird dann auch manchmal als ubiquitäres System bezeichnet [31]. Innerhalb dieser Infrastruktur sind dann verschiedene Interaktionsformen denkbar, die es erlauben, die angebotenen Ressourcen optimal zu nutzen. Die von Weiser später in *Calm Technology* [35] umgetaufte unbewusste Nutzung ist dabei nur noch eine von vielen Möglichkeiten. Meist wird jedoch keine bestimmte Strategie oder Technik favorisiert, sondern eine Mischung von dem jeweiligen Zweck angepassten expliziten und impliziten Interaktionsmöglichkeiten (*Appropriate Computing*) [31]. Diese letztgenannte Deutung kommt der situativen Definition (Def. 1) des Begriffes aus technischer Sicht bereits sehr nahe.

Entwicklung zum Sammelbegriff durch Abstraktion der unbewussten Interaktionsfom.

Leider herrscht aber auch über die genaue Zusammensetzung (viele statische vs. einige tragbare Geräte) des infrastrukturellen Rahmens keine begrifflich geschlossene Einigkeit. Aus diesem Grund wurde für diese Arbeit eine Definition von Ubiquitous Computing gewählt, die sowohl von der Art der Interaktion als auch von der Hardware abstrahiert und lediglich das Ziel einer vermehrten Computernutzung in möglichst vielen Situationen formuliert. Damit sind dann sowohl tragbare Geräte, als auch vernetzte Umgebungen genauso wie eine bewusste oder unbewusste Nutzung darin subsumiert.

1.1.2 Abgrenzung zu anderen Begriffen

Abstraktion von Art der Realisierung und Interaktion.

Für die situative Definition von Ubiquitous Computing war eine Abstraktion hin auf eine Zielvorstellung der Mehrbenutzung von Computern, unabhängig von der Art der Interaktion und technischen Realisierung verwendet worden. Ubiquitäre Anwendungen werden damit zu einer übergeordneten Einheit, die unabhängig von einer konkreten HW/SW- Implementierung existieren können muss.

Diese besondere Art der Definition leistet zwar die gewünschte Vereinheitlichung der unterschiedlichen Aspekte und lösungsorientierten Definitionen, erschwert aber gleichzeitig auch eine klare Abgrenzung zu anderen Begriffen, die sich bezüglich der situativen Definition überschneiden. Beispiele dafür sind etwa das *Mobile-*, *Nomadic-* oder *Wearable Computing* (siehe auch Abschnitt A1.1 im Anhang). Ein geeignetes Abgrenzungskriterium der situativen Definition ist in solchen Fällen die *Nutzbarkeit*.

Definition 4: *nutzbar* _____

Eine Anwendung ist einer bestimmten Situation nutzbar, wenn:

- die notwendige Infrastruktur aus Hard und Software in dieser Situation für die Anwendung *verfügbar* gemacht werden kann,
- die Anwendung die momentanen Benutzerbedürfnisse erfüllen kann (d.h. *anwendbar* ist) und
- die dafür notwendigen Interaktionen nicht mit der Benutzersituation in Konflikt stehen, also den momentan freien Interaktionsfähigkeiten entsprechen. Die Anwendung ist somit im weitesten Sinne (d.h. bis hin zu einer vollständigen Automatisierung) *bedienbar*.

Der wichtigste Unterschied zwischen Ubiquitous- und beispielsweise Mobile Computing ist also, dass bei Ersterem nicht mehr lediglich die statische Überall-Verfügbarkeit von Hardware, Daten oder Diensten im Vordergrund steht,

sondern deren tatsächliche *Nutzbarkeit* und damit implizit auch die Herstellung einer temporären (dynamischen) *Verfügbarkeit*. Dass sich *Verfügbarkeit* und *Nutzbarkeit* unterscheiden, wird schnell an einem Beispiel deutlich.

> Die reine Verfügbarkeit eines Fernsehers im Auto erlaubt nicht zwingend auch seine Nutzung. Diese hängt vielmehr wesentlich von der Situation der im Auto anwesenden Personen ab.
>
> Ist der Fernseher dagegen gerade nicht nutzbar dann kann seine statische Verfügbarkeit darüber hinaus möglicherweise sogar schädlich für die Nutzbarkeit anderer Funktionen sein (Stromverbrauch, Platzangebot etc.).
>
> In solchen Fällen kann es daher vorteilhaft sein, die für das Fernsehen notwendigen Ressourcen lediglich bei Bedarf dynamisch verfügbar zu machen, beispielsweise durch die gemeinsame Nutzung des Anzeigegerätes als Instrumententafel, Navigationssystem oder Medienwiedergabe.

Nutzbarkeit als Abgrenzungskriterium

Nicht jede mobile Anwendung ist folglich auch ubiquitär. Eine ubiquitäre Anwendung kann umgekehrt jedoch durchaus mobil sein, um die für die Nutzbarkeit notwendige Verfügbarkeit sicherzustellen. Das oben genannte Beispiel von Anwendungsnutzung im Auto zeigt aber auch, dass eine lediglich statische Überall-Verfügbarkeit aller Informationen und Dienste keinesfalls in jedem Fall als Ideal anzustreben ist, da das bestenfalls unnütz ist. Schlechtestenfalls kann sich die Nutzbarkeit wegen der Unübersichtlichkeit, Ablenkung des Benutzers oder zu hohem Ressourcenverbrauch sogar verschlechtern.

Die Behauptung, die Betonung der Nutzbarkeit sei das wesentliche Merkmal des Ubiquitous Computing kann auch mithilfe einiger anderer Überlegungen validiert werden (siehe Anhang A1). Aus der situativen Definition kann dazu wie bereits erwähnt ein mehrdimensionaler Eigenschaftsraum mit den Hauptdimensionen "Verfügbarkeit" und "Nutzbarkeit" konstruiert werden. In den davon aufgespannten Raum lassen sich die verschiedenen Ausprägungen des Ubiquitous Computing und auch der Definitionen verwandter oder enthaltener Begriffe wie dem Mobile Computing anhand ihrer Eigenschaften einordnen. Dies zeigt zum einen, dass die allgemeine Definition des Begriffes einerseits auch wirklich die verschiedenen lösungsorientierten Sichtweisen auf den Begriff enthält. Zum anderen wird ein Bereich des Eigenschaftsraumes sichtbar, der sich nicht mit anderen bisher bekannten Nutzungsformen von Computern überschneidet. Dieser Aspekt des Ubiquitous Computing beschreibt die weitgehend automatisierte Interaktion zwischen Nutzer und Anwendung beziehungsweise HW/SW-Systemkomponenten untereinander

in einer stark heterogenen und veränderlichen Umgebung. Dies entspricht genau dem wesentlichen Unterscheidungsmerkmal der angepassten Nutzbarkeit in unterschiedlichen Situationen. Als Schlussfolgerung kann man daher ziehen, dass die die Ubiquität bestimmtende Eigenschaft aus einer situationsabhängigen HW/SW-Realisierung einer situations- und benutzerspezifischen Anwendung und Interaktionsform besteht. Dadurch wird gegenüber bisherigen Nutzungsformen die Nutzung von Computern und Computeranwendungen auf die maximal mögliche Anzahl von Situationen eines Benutzers ausgeweitet. Damit ist aber auch gleichzeitig eine Verknüpfung des wesentlichen Merkmals des Ubiquitous Computing mit der Adaption als einer wesentlichen Eigenschaft ubiquitärer Anwendungen hergestellt. Mark Weisers ursprüngliche Idee der unbewussten oder Hintergrundnutzung spielt dabei noch immer eine große Rolle. Allerdings bezieht sich diese Rolle im Allgemeinen nur noch auf die Adaption selbst und den Teil der angepassten Anwendungsfunktionen, für den die Benutzerbedürfnisse unbewusste Nutzung in einer bestimmten Situation gerade erfordern, nicht mehr auf das gesamte System.

Automatische Adaption an Situation (und Nutzer) ist wesentliche Eigenschaft.

Die automatische Übertragung und damit Adaption eines gerade noch mobil auf einem Smartphone betrachteten Informationsdienstes auf einen nach Betreten eines Raumes verfügbaren stationären Bildschirm findet im Hintergrund statt. Die Nutzung des Dienstes kann aber durchaus die direkte und bewusste Aufmerksamkeit des Benutzers erfordern. Trotzdem kann durch die Adaption die Nutzbarkeit erhöht, oder aufgrund der größeren Betrachtungsfläche überhaupt erst hergestellt werden.

Der Bezug zu den Benutzerbedürfnissen hilft auch bei einer Abgrenzung des Ubiquitous Computing gegenüber autonomen Systemen in der Robotik oder im Bereich der Softwareagenten. Diese selbstständigen Systeme erfüllen oft nur eine oder mehrere Aufgaben in möglichst vielen Situationen des Systems. Lediglich wenn eine dieser Aufgaben darin besteht, die Benutzerbedürfnisse mit unterschiedlichen zur Verfügung stehenden Mitteln zu erfüllen (und zu erkennen), wird daraus eine ubiquitäre Anwendung nach Definition 1.

1.1.3 Vorteile von Ubiquitous Computing

Technischem Aufwand der Adaption stehen entsprechende Vorteile gegenüber

Die weitgehende Unabhängigkeit von der technischen Realisierung und der Anspruch, sich situationsbedingt an technische Erfordernisse und die Bedürfnisse des Benutzers anzupassen, legen nahe, dass ubiquitäre Anwendungen mit einer Reihe von technischen und methodischen Herausforderungen konfrontiert sind. Dieser Aufwand muss natürlich auf einer anderen Seite durch entsprechende Vorteile kompensiert werden.

Im Falle des Ubiquitous Computing sind die Hauptvorteile offensichtlich eine aus der Nutzbarkeitsdimension ableitbare höhere Nutzungsintensität und ein besserer Anwendungskomfort. Zum Beispiel lassen sich ubiquitäre Anwendungen auch in Nutzungssituationen verwenden, die bisher weitgehend von einer Computernutzung ausgenommen waren. Dazu zählen fast alle Alltagssituationen, in denen die Aufmerksamkeit des Benutzers ganz oder teilweise auf andere Tätigkeiten gerichtet ist (Auto fahren, soziale Interaktion, Unterhaltung etc.). Daneben ergeben sich aber noch weitere Vorteile aus der für die Nutzbarkeit notwendigen Dimension der dynamischen Verfügbarkeit:

Vorteil 1: Ausdehnung von Computernutzung auf Alltagssituationen

- Höhere Effizienz durch Einsparung redundanter Ressourcen.
  Die Erkennung von Benutzerbedürfnissen erlaubt neben der Verbesserung der Bedienbarkeit eine bedarfsorientierte Steuerung dynamischer Ressourcenverfügbarkeit. In nichtubiquitären Anwendungen werden wichtige Ressourcen dagegen in Form einer statischen Superposition auch dann bereitgehalten, wenn sie nicht genutzt werden, weil über die zukünftige Nutzungswahrscheinlichkeit keine Annahmen gemacht werden können. In Nutzungsszenarien, die mehrere unabhängige Teilsysteme kombinieren, müssen dadurch viele Ressourcen redundant ausgelegt sein (Rechenleistung, Bildschirme, Tastaturen etc.).

Vorteil 2: Bessere Ausnutzung veränderlicher Ressourcentuationen

- Größter gemeinsamer Funktionsumfang statt kleinster gemeinsamer Nenner.
  Dynamische Erkennung und Einbindung von Ressourcen und Funktionen erlaubt die volle Ausnutzung von unabhängigen und teils heterogenen Teilsystemen in mobilen Anwendungsszenarien oder anderen Fällen stark variierender Systemumgebungen (z.B. durch hohe Ausfallwahrscheinlichkeit).
- Einfache und nachträgliche Erweiterbarkeit
  Die Steuerung dynamischer Verfügbarkeit erlaubt auch das nachträgliche Verändern oder Erweitern bestehender Systeminstanzen zur Laufzeit. Die Änderbarkeit zur Laufzeit ist auch für Anwendungen in stabilen Systemumgebungen als Vorteil geeignet, sofern es sich um Hochverfügbarkeitsanwendungen handelt, die für Wartungsarbeiten nicht ohne weiteres deaktiviert werden können.

Vorteil 3: Erhöhung von Lebens- und Betriebsdauer

1.1.4 Beispiele für Ubiquität in Softwaresystemen

In der Softwaretechnik ist das Konzept der Ubiquität bereits anhand von einigen, meist sehr spezialisierten Anwendungen untersucht worden. Diese Bei-

spiele können als Illustration für die gerade genannten Vorteile herangezogen werden.

Der *CybreMinder* [21] basiert beispielsweise auf einer einfachen Aufgabenliste, wie sie aus verschiedenen PIM Anwendungen wie beispielsweise Microsoft Outlook her bekannt ist. Im Gegensatz zur Nutzung herkömmlicher Aufgabenlisten, die in der Regel nur das Eintragen einer Aufgabe erlauben, an deren Erfüllung zu einem bestimmten Zeitpunkt erinnert wird, ermöglicht der CybreMinder die Verknüpfung einer Aufgabe mit beliebigen anderen Situationen. Auf diese Weise kann an eine Aufgabe auch früher erinnert werden, wenn gerade nichts zu tun ist, oder die Erinnerungsfunktion kann daran geknüpft werden, dass ein bestimmter Ort besucht wird, oder eine andere Person sich an einem bestimmten Ort aufhält und beispielsweise deshalb einen anstehenden Telefonanruf gerade gut annehmen könnte.

The Garden Workspace [22] ist ein Szenario, das am MIT konzipiert und realisiert wurde. Es handelt sich dabei um einen gemeinschaftlichen Aufenthalts- und Durchgangsbereich des MIT Media Labors. Darin befindet sich eine große Projektionsfläche auf der die verschiedensten Informationen dargestellt werden, beispielsweise Artikelüberschriften von Nachrichten. Vorbeigehende Personen werden von einer Kamera erfasst. Bleibt ein Benutzer stehen, wird der zur gerade eingeblendeten Überschrift gehörende Artikel abgerufen und eingeblendet. Verweilt die betreffende Person noch länger, werden verwandte Artikel zu ähnlichen Themen angezeigt. Welche Artikel gelesen werden, wird aufgezeichnet und dafür verwendet, später die Auswahl der angezeigten Informationen zu verbessern.

An der Technischen Universität München (TUM) wurde mit *Mobi@* ein Szenario und ein Konzept für eine kontextabhängige Suchmaschine für den mobilen Einsatz entwickelt [104] und patentiert [49]. Der Benutzer verwendet im Szenario sein normales Mobiltelefon. Das System stellt automatisch seinen Standort fest und sucht zum Beispiel nach allen Restaurants im Umkreis, die geöffnet haben und den Vorlieben des Benutzers entsprechen, zum Beispiel Schnellrestaurants mit vegetarischen Gerichten (Abb. 6 Nr. 1). Dabei kann der Benutzer Verweise zu gefundenen Informationen abspeichern, bewerten und später wieder abrufen (Nr. 2). Auf diese Weise werden reale Orte mit weiteren informationen versehen, die im System gespeichert und für weitere Auswertungen herangezogen werden können.

Das Ziel von Mobi@ ist, dass der Benutzer relevante Daten und möglicherweise nutzbare Dienste angezeigt bekommt, die zu seiner jeweiligen Situation und seinen Vorlieben passen, ohne dass er eine explizite Anfrage stellen muss. Im Szenario sind das die Wegbeschreibung zum Restaurant, eine Unwetterwarnung, und die Möglichkeit, noch während dem Essen ein Taxi zu bestel-

len, um trockenen Fußes sein Hotel zu erreichen (Nr. 3-5). Die Suchmaschine verändert sich dadurch zu einer Findmaschine, präzisiert durch Einbeziehung von Zusatzinformationen wie Ort, Zeit, Vorlieben des jeweiligen Benutzers und sein früheres Verhalten, bzw. das Verhalten von anderen Personen mit ähnlichen Vorlieben in ähnlichen Situationen. Das Beispiel illustriert, wie die Berücksichtigung von Situationen zu einer Ausrichtung auf die Bedürfnisse des Benutzers führt.

Abb. 6: Mobi@ - Situationsspezifische Auswahl von ortsbasierten Services

1.2 Motivation dieser Arbeit

Gerade im Zusammenhang mit intensiven und kritischeren Diskussionen über den praktischen Nutzen des Ubiquitous Computing werden des Öfteren fachliche oder experimentell begründete Zweifel an der generellen Tauglichkeit des Konzeptes geäußert.

Dies beginnt bereits mit dem Begründer des Begriffes selbst, der die Feststellung äußerte, dass die meisten prototypischen Versuche, Computernutzung vollständig in die periphere Hintergrundwahrnehmung zu verlagern, erfolglos waren [8]. Auch objektive Berichterstattungen über Langzeitexperimente, beispielsweise die Bewohner eines experimentellen Hauses [69] oder breiter angelegte Studien [64] berichten über Probleme, wenn Anwendungen mit dem Anspruch antreten, in heterogenen beziehungsweise dynamischen Umgebungen zu agieren oder die Bedürfnisse des Benutzers von sich aus erken-

nen und erfüllen zu können und damit überall und jederzeit (auch im Hintergrund) nutzbar also ubiquitär zu sein.

Spontanes Fehlverhalten als größtes konzeptionelles Problem des Ubiquitous Computing

Solche Bedenken konnten auch für Prototypen bestätigt werden, die im Vorfeld dieser Arbeit am Kompetenzzentrum für Mobile & Context Aware Computing des Lehrstuhls für Software und Systems Engineering der TUM entwickelt wurden. Diese zeigten unter realitätsnahen Bedingungen häufig eine spezielle Art von spontanem Fehlverhalten, obwohl im Labor alle Tests erfolgreich absolviert worden waren. Da das Auftreten des Problems einen schwer vorherzusehenden phänomenartigen Charakter besaß und dafür noch kein eindeutig identifizierender Begriff existierte, wurde es in Anlehnung an seinen, die Anwendbarkeit von Ubiquitous Computing untergrabenden (subversiven) Charakter, als *Spontaneous Unexpected Behavior* (SUB) bezeichnet.

Gerade das spontane Auftreten grenzt das SUB-Phänomen gegenüber klassischen Realisierungs- oder Spezifikationsfehlern ab. Es handelt sich bei SUB vielmehr um ein temporäres oder stetiges Divergieren von in der Anwendung statisch realisierten Anforderungen und sich weiterentwickelnden dynamischen Anforderungen des Benutzers. Das Muster ist dabei immer gleich. Eine Anwendung funktioniert für einen bestimmten Zeitraum zur vollsten Zufriedenheit (d.h. gemäß den Anforderungen) der Benutzer. Nach längerem Betrieb oder intensiverer Nutzung tritt jedoch eine neue Situation auf, in der das Systemverhalten in deutlicher Weise von den Nutzererwartungen abweicht (unerwartetes Verhalten, divergierende Nutzeranforderungen). Eine Analyse solcher Ereignisse zeigte dann jeweils, dass weder Fehler in der Realisierung noch falsche Anforderungen der Grund waren. Ursache waren vielmehr bei der Modellierung der Anforderungen oder des technischen Systems getroffene und zu diesem Zeitpunkt richtige beziehungsweise vernünftige Annahmen. Zum Beispiel, welche Kriterien für die Erkennung von Benutzerbedürfnissen herangezogen werden oder welche technische Infrastruktur zur Verfügung stehen könnte. Solche Annahmen schließen häufig zum Entwicklungszeitpunkt unwirtschaftliche oder unvorhersehbare Sonderfälle explizit oder implizit aus. Gerade in ubiquitären Anwendungen sind dem Benutzer die so realisierten Anforderungen und ihre Grenzen aber in der Regel nicht bewusst. Tritt dann eine durch die Modellierungsannahmen ausgeschlossene kritische Situation dennoch ein, kommt es in Folge zu einem überraschenden und meist bezogen auf die Nutzerbedürfnisse fehlerhaften Verhalten der Anwendung aus Sicht und Wertung des Benutzers. SUB Phänomene entstehen also grob vereinfacht durch ein Missverständnis zwischen dem Entwickler (vertreten durch das System) und Anwender über die Einschätzung einer Situation oder über die Fähigkeiten und Grenzen der Applikation.

Dieses Phänomen ist natürlich offensichtlich nicht auf ubiquitäre Systeme beschränkt, wie das das berühmte Beispiel des "Jahr 2000" (Y2K) Phänomens gezeigt hat. Einige der davon betroffenen Programme waren nicht fehlerhaft im Sinne menschlichen Versagens, sondern enthielten lediglich die (zum Entwicklungszeitpunkt vernünftige und richtige) implizite Annahme, dass das betroffene Programm nur für einen bestimmten Zeitraum im Einsatz sein würde, für den eine Reduzierung der Datumsangaben auf das Jahrzehnt keine Probleme bereiten würde. Die Programme waren also zum Zeitpunkt der Auslieferung und über lange Zeit ihres Betriebes in diesem Sinne fehlerfrei, sofern nicht eine anders lautende explizite Anforderung bestand. Dennoch konnten sie zu einem späteren Zeitpunkt ein fehlerhaftes Verhalten zeigen, insbesondere wenn dem Benutzer diese Einschränkung nicht bewusst war.

In ubiquitären Anwendungen treten vergleichbare Situationen jedoch offensichtlich gehäuft (oder zumindest merklicher) auf. Dies geht bis hin zu einem Punkt, an dem die ubiquitären Systeme den ursprünglichen Anspruch, die Benutzerbedürfnisse zu erfüllen, über einen längeren Zeitraum hinweg nicht aufrechterhalten können. Diesen Umstand nehmen zunehmend mehr kritische Publikation zum Anlass, ubiquitären Anwendungen nicht nur technische, sondern vor allem auch konzeptionelle Schwächen zu unterstellen [23][70].

Als Begründung dieser Generalkritik wird meist eine Verbindung zur Künstlichen Intelligenz (KI)-Forschung hergestellt. Für die Erkennung von Benutzerbedürfnissen ist es nämlich notwendig, eine große Menge von Umgebungsinformationen in einen größeren Zusammenhang einzuordnen. So müsste etwa der Kühlschrank als Teil einer im Hintergrund genutzten ubiquitären Anwendung erkennen, dass die in ihm gelagerten teuren Feinkostvorräte nur für eine Feier gedacht sind und nicht ständig nachbestellt werden müssen [70]. Im Bezug auf diese Aufgabe ist in der KI-Forschung allerdings in den letzten 20 Jahren kein nennenswerter Durchbruch gelungen. Stattdessen ist das Problem zu einer prinzipiellen Problematik (*Frame-Problem* [66]) erklärt worden, deren grundsätzliche Lösbarkeit außerhalb von Einzelfällen stark eingeschränkter Modellierungsgegenstände noch immer in Frage gestellt wird.

Im Bezug auf Ubiquitous Computing wird diese Problematik damit aber zum Kernproblem und seine Lös- oder Umgehbarkeit zur Antwort auf die Frage nach dem praktischen Nutzen von Ubiquitous Computing, den Marc Weiser ja indirekt durch sein Scheitern in Frage gestellt hatte. Ziel dieser Arbeit ist es daher, die konzeptionellen Schwächen bezüglich des Frame-Problems im Entwurf und bei der Realisierung ubiquitärer Anwendungen zu beseitigen.

Dabei steht naturgemäß nicht die Lösung eines 20 Jahre alten und mit großer Wahrscheinlichkeit unlösbaren Problems im Vordergrund, sondern vielmehr dessen Vermeidung, Abmilderung (oder Umgehung).

Die Grundüberlegung zu dieser Herangehensweise stützt sich darauf, dass das Frame-Problem prinzipiell auch in nichtubiquitären Anwendungen auftreten kann. Dennoch spielt es dort in der Regel keine wichtige Rolle. Gelingt es daher, einen Zusammenhang zwischen typischen Entwurfs- und Realisierungstechniken für ubiquitäre Systeme und der zunehmenden Bedeutung des Frame-Problems herzustellen, lassen sich diese Techniken im Entwurfsprozess zum Teil vermeiden oder ersetzen.

Handelt es sich dagegen um einen inhärenten Zusammenhang zwischen der Definition von Ubiquität (Anspruch der Nutzbarkeit in unterschiedlichen Situationen) und dem Frame-Problem, dann gelingt zumindest konzeptionell eine Vermeidung der Problematik durch eine Erkennung und Vermeidung der kritischen Situationen. Am Beispiel des Kühlschranks bedeutet dies etwa, dass der Nutzer über dessen Funktionsweise Bescheid wissen könnte und aufgrund dieser Tatsache die kritische Situation im Vorfeld erkennen und verhindern kann, indem er die automatische Inventar- und Bestellfunktion vorübergehend deaktiviert, solange sich die unüblichen Vorräte im Kühlschrank befinden, also die Ausnahmesituation anhält.

1.3 Struktur und Ergebnisse

Um die Ergebnisse der Frage nach der besten Strategie zur Vermeidung des Frame-Problems in ubiquitären Systemen vorwegzunehmen; beide genannten Vermeidungsstrategien haben ihre Berechtigung, allerdings handelt es sich bei dem Zusammenhang zwischen verstärktem Auftreten des Frame-Problems und Ubiquität tatsächlich (zum großen Teil) um eine inhärente Eigenschaft ubiquitärer Anwendungen. Dadurch wäre es notwendig im Sinne der ersten Strategie ganze Gruppen von Anwendungsfällen zu vermeiden, um eine Reduzierung der Auswirkungen des Frame-Problems zu erzielen.

Die vorliegende Arbeit konzentriert sich folglich auf die zweite Vermeidungsstrategie, bei der versucht wird, kritische Situationen zur Laufzeit zu erkennen und zu beseitigen, nötigenfalls auch durch zeitweises Deaktivieren oder Reprogrammieren bestimmter Funktionen.

Dazu wird zunächst in einem ersten Schritt das Ergebnis der Analyse des genauen und auch technischen Zusammenhangs zwischen Ubiquität und Frame-Problem dargelegt (Kapitel 3):

- Das Frame-Problem basiert neben perspektivischen auch auf teilweise zeitlich bedingten Divergenzen zwischen Modellierungen und der Wirklichkeit. Da sich ubiquitäre Systeme wegen ihrer Adaptionsfähigkeit durch höhere Komplexität und längere Betriebsdauer auszeichnen, steigt damit die Anfälligkeit für das Frame-Problem.
- Viele Anwendungsfälle des Ubiquitous Computing haben einen direkteren Einfluss auf die Umgebung des Nutzers (im Vergleich zu einem reinen Infoterminal). Dadurch steigt die Perzeption (der Störfaktor) von nichtkonformem, das heißt nicht den Nutzererwartungen entsprechendem Verhalten des Systems. Gleiches gilt für einen direkt formulierten Anspruch, die Benutzerbedürfnisse zu erkennen und zu erfüllen. Die Anwendung wird damit vom richtig oder falsch eingesetzten Werkzeug zum guten oder schlechten Dienstleister.
- Die Eigenschaft ubiquitärer Anwendungen, Nutzbarkeit auch für Situationen mit beschränkten Interaktionsmöglichkeiten oder Fähigkeiten des Benutzers herzustellen (durch Verlagerung in den Hintergrund der Benutzerwahrnehmung und Automatisierung) reduziert die Möglichkeiten des Benutzers, das Frame-Problem von außen zu kompensieren, wie das in nichtubiquitären Anwendungen der Fall ist. Kurz gesagt passt sich im ubiquitären Fall der Benutzer nicht mehr länger an die Eigenheiten des Programms an, um dessen Defizite zu kompensieren.

Genau im letzten Punkt setzt aber die in dieser Arbeit fokussierte Strategie zur Vermeidung des Frame-Problems an. Diese besteht darin, ein Verfahren mit der Bezeichnung *Kalibrierung* zu konzipieren, mithilfe dessen die verloren gegangene Kompensationsmöglichkeit ersetzt werden kann, ohne die Ubiquität zu zerstören.

Kern des Verfahrens ist eine explizite Modellierung und Realisierung von kontextadaptivem Verhalten (beispielsweise wie sich das System an die Benutzerbedürfnisse anpasst). Ein konkretes Modell eines Anpassungsverhaltens (K-Modell) kann mithilfe eines geeigneten Framework (Kapitel 6) realisiert werden. Das Framework erlaubt wiederum eine Methodik (Kapitel 7) anzugeben, mithilfe derer im Entwurf für eine konkrete ubiquitäre Anwendung eine geeignete Kalibrierung konstruiert werden kann. Geeignet bedeutet in diesem Zusammenhang, dass die Kalibrierung in der Lage ist, für eine konkrete Anwendung das Frame-Problem so weit zu kompensieren, wie das in nichtubiquitären Systemen durch die interaktive Benutzerschnittstelle geschehen kann. Allerdings ohne durch deren ubiquitätsbeschränkende Nachteile in der Nutzbarkeit eingeschränkt zu werden. Dadurch wird das Ziel erreicht, die negativen Auswirkungen des Frame-Problems durch seine

Vermeidung auch in ubiquitären Anwendungen auf ein akzeptables Maß zu senken und damit eine praxisgerechte Anwendbarkeit zu garantieren.

1.3.1 Vergleichbare Arbeiten

Vergleichbare Arbeiten auf diesem Gebiet lassen sich vor allem in drei große Kategorien einordnen:

- Fallstudien und Referenzarchitekturen für Kontext.
  In diese Kategorie fallen die meisten Arbeiten zum Thema Kontext und *Context Awareness*, beispielsweise [26], [28], [29]. Sie basieren jeweils auf einem informellen Konzept zur Verarbeitung von Kontext, einer der Konzeption entsprechenden Referenzarchitektur und ein oder mehreren Beispielanwendungen.
  Hauptsächliche Unterscheidungsmerkmale zwischen den verwandten Arbeiten dieser Kategorie und der vorliegenden Arbeit sind das formale Modell der Adaption, die Behandlung von SUB-Phänomenen und ihrer Kalibrierung sowie die Entwicklung verschiedener methodischer Entwurfstechniken rund um die Kontextadaption.
- Multimodale Benutzerschnittstellen
  Arbeiten über multimodale oder alternative Interaktionstechniken beschäftigen sich mit neuen benutzerfreundlichen Interaktionstechnologien wie Sprache, Gestik, Mimik oder Kombinationen davon. Kontextadaption verfolgt dagegen einen allgemeineren Ansatz der Betrachtung von Interaktionen zwischen Systemen und ihrer Umgebung. Bei der Umgebung kann, muss es sich aber nicht zwangsläufig um einen menschlichen Benutzer handeln. Diese Arbeit unterscheidet sich von verwandten Arbeiten über multimodale oder alternative Benutzerschnittstellen ebenfalls dadurch, dass per se nicht zwischen Wünschen des Benutzers und seiner Situation (welche die Bedürfnisse des Benutzers umfasst) unterschieden wird. Zudem wird als Neuerung die Kalibrierung als dritte Form der Interaktion neben direkter Bedienung und Kontextadaption eingeführt. Diese dritte Interaktionsform kann als Eichung beziehungsweise Personalisierung einer Systeminstanz auf die spezielle Einsatzsituation verstanden werden.
- Workflow- und Prozessbeschreibungssprachen
  Diese und vergleichbare Konstruktionskonzepte ähneln den Konzepten der Kontextadaption und Kalibrierung dahingehend, dass versucht wird, aus einer vorgegebenen Menge von Hardware und Softwarekom-

ponentenressourcen maßgeschneiderte Lösungen für bestimmte Einsatzsituationen zu konstruieren. Im Unterschied zur Adaption erfolgen die Anpassungen jedoch immer manuell durch einen Entwickler. Die Kalibrierung basiert hingegen zwar letzten Endes auch immer auf einer bewussten Interaktion mit einem Menschen, es handelt sich dabei aber üblicherweise um den Endbenutzer der Anwendung. Zudem können Adaption und Kalibrierung zur Laufzeit eines Systems angewandt werden. Verantwortlich dafür ist die für Adaption geforderte lose Kopplung der jeweils eigenständige Prozesse darstellenden Komponenten über eine zustandsbehaftete Zwischenschicht (den Kontext).

1.4 Zusammenfassung der Ziele und Ergebnisse

Die einleitenden Aussagen des ersten Kapitels noch einmal zusammengefasst, bestehen die Ziele dieser Arbeit darin:

- Die Praxistauglichkeit ubiquitärer Anwendungen durch Reduzierung des SUB-Phänomens zu verbessern.
- SUB mithilfe des Frame-Problems zu erklären.
- Ein Konzept (genannt Kalibrierung) zu entwickeln, wie das Frame-Problem in ubiquitären Anwendungen vermieden oder umgangen werden kann.
- Das Konzept technisch (Framework) und methodisch (Entwurf) zu realisieren

Die in diesem Zusammenhang erzielten Ergebnisse bestehen aus:

- Einer Analyse des Zusammenhangs zwischen SUB und Frame-Problem.
- Einer formalen Modellierung von Kontextadaption (K-Modell).
- Der Spezifikation eines Framework, das die Realisierung einer Kalibrierung (Veränderung zur Laufzeit) einer so modellierten Kontextadaption (K-Modell) ermöglicht.
- Einer Methodik für den Umgang mit K-Modellen und ihrer Laufzeitkalibrierung im Entwurf auf Basis des zuvor spezifizierten Framework.

Die Struktur der Arbeit gliedert sich gemäß den oben genannten Ergebnissen. Zuvor wird allerdings im folgenden Kapitel eine Fallstudie vorgestellt, die im weiteren Verlauf der Arbeit als Quelle für praktische Beispiele dient.

Darüber hinaus enthält Kapitel 2 zur Vertiefung der Problematik eine Übersicht über eine Reihe von SUB-Phänomenen aus der Praxis.

KAPITEL 2 Fallstudie Smart Home

Wie sieht eine typische ubiquitäre Anwendung aus und welche SUB-Phänomene können darin auftreten?

Das "Intelligente Haus" als typisches Beispiel von Ubiquitous Computing

Dieses Kapitel beschreibt einen konkreten Anwendungsfall von Ubiquitous Computing anhand eines so genannten "Intelligenten Hauses". Unter dieser Bezeichnung versteht man allgemein die Integration von softwarebasierter Regelungstechnik aus der Gebäudeautomation mit einem über mehrere unterschiedliche Geräte hinweg verteilten Informationssystem. Die verwendeten Regelungsmechanismen und Interaktionsmöglichkeiten zwischen System und Benutzer können dabei sehr aufwändige Formen annehmen, bis hin zur Verwendung von Methoden der künstlichen Intelligenz (KI). Das Verhalten eines solchen Systems kann dadurch zeitweise von seinem Benutzer bei oberflächlicher Betrachtung als intelligent empfunden werden (vgl. *Turing Test* u.ä. [84]). Daraus ergibt sich der Begriff "Intelligentes Haus" und synonyme Bezeichner wie *Intelligent Home*, *Smart Home* oder *eHome*. Es handelt sich dabei um ein typisches Ubiquitous Computing Szenario, da die Nutzbarkeit der Anwendung in unterschiedlichen Situationen im Vordergrund steht. Diese können einerseits aus verschiedenen Alltagssituationen des Benutzers bestehen, die zum Teil eine indirekte oder sogar unbewusste Interaktion erfordern. Andererseits können sich auch unterschiedliche HW/SW-Situationen ergeben, bedingt durch eine variierende Verfügbarkeit technischer Ressourcen in verschiedenen Haushalten, die sich zudem durch mobile oder neu hinzugekaufte Geräte dynamisch zur Laufzeit verändern können. Die folgende Fallstudie dient begleitend zu dieser Arbeit als durchgängiges Beispiel für ubiquitäre Anwendungen und gleichzeitig als eine Möglichkeit, SUB-Phänomene und ihre Vermeidung in der Praxis zu studieren.

Im ersten Abschnitt dieses Kapitels (2.1) wird das untersuchte Anwendungsgebiet zunächst durch ein kleines Szenario verdeutlicht, das als Grundlage für

alle weiteren behandelten Beispiele dient. Teile dieses Szenarios wurden innerhalb von zwei aufeinander folgenden Prototypen realisiert. Dabei bediente sich der erste Prototyp Konzepten und Architekturen, welche dem Stand der Kunst (*state of the art*) zu Beginn der Fallstudie entsprachen. Der zweite Prototyp basiert dagegen auf den Ergebnissen der vorliegenden Arbeit, insbesondere auf dem in Kapitel 6 beschriebenem Framework. Im zweiten Abschnitt dieses Kapitels (2.2) folgt eine kurze Aufstellung der prototypisch realisierten Funktionalitäten. Abschnitt 2.3 beschreibt dann den genauen Aufbau des ersten Prototyp und Abschnitt 2.4 zeigt, welche verschiedenen Ausprägungen des SUB-Phänomens sich nach dessen längerem Betrieb feststellen ließen.

Komplettiert wird die Fallstudie durch die Beschreibung eines zweiten Prototyp (II) in Kapitel 7 in Form eines durchgängigen Entwurfsbeispiels. Dieser zweite Prototyp entspricht bezüglich der Funktionalität im Wesentlichen dem ersten. Allerdings wurde er basierend auf den Ergebnissen dieser Arbeit unter anderem mithilfe des Framework aus Kapitel 6 neu realisiert und um einige Kalibrierungsmechanismen erweitert. Auf diese Weise konnten die in den nächsten beiden Kapiteln aufgestellten Anforderungen an einen Kalibrierungsmechanismus, der dazu geeignet ist, die Ursachen von bereits aufgetretenen oder absehbaren SUBs zur Laufzeit zu vermeiden, validiert werden. Das Ergebnis der Validierung wird später in Kapitel 8 noch einmal zusammengefasst.

2.1 Szenario

Als Ausgangspunkt der in dieser Arbeit begleitend eingesetzten Fallstudie wurde ein "Intelligent Home" Szenario aus dem Umfeld der Gebäudeautomation ausgewählt. Diese besonders im Bereich privater Wohnräume angesiedelten Szenarien stellen eine Unterklasse von Smart Spaces [93] und damit Anwendungen des Ubiquitous Computing dar, bei denen verschiedenartigste Funktionen der Informations- und Regelungstechnik über eine größere Anzahl von miteinander vernetzten und teilweise mobilen Geräten hinweg verteilt ist. Bei der Auswahl eines solchen Szenarios als Grundlage der Fallstudie spielten folgende Überlegungen eine Rolle:

- Intelligente Häuser sind die einzigen konkreten Beispiele komplexer ubiquitärer Systeme, für die eine größere Anzahl voneinander unabhängiger Studien der Praxistauglichkeit unter realen Bedingungen existiert (z.B.: [85],[24],[25],[86])

- Es ist mit vertretbarem Aufwand möglich, Teile eines solchen Szenarios in einem praktischen Versuchsaufbau zu realisieren und unter praxisnahen Bedingungen auf bestimmte Effekte hin zu untersuchen.
- Szenarien in Zusammenhang mit Computeranwendungen in Wohnräumen zeigen sich bezüglich Heterogenität, Lebensdauer und Störfaktoren als besonders anfällig gegenüber dem zu untersuchenden Phänomen.

Grund für die bessere und auch kritischere Erforschung von Szenarien aus dem Bereich der Heimautomation ist, dass diesen (neben vergleichbaren Anwendungen aus dem Automobilbereich) als einzigen ubiquitären Anwendungen für die nähere Zukunft eine begrenzte Marktrelevanz vorausgesagt und damit auch eine gewisse öffentliche Aufmerksamkeit zuteil wird.

Einfache Realisierung und Möglichkeit zur Langzeitbeobachtung

Für die Eignung als Grundlage einer Fallstudie ist aber ebenso von erheblichem Belang, dass ein entsprechender Prototyp mit vertretbarem Aufwand realisiert und unter praxisnahen Bedingungen (z.B. in den eigenen vier Wänden) über einen längeren Zeitraum hinweg beobachtet werden kann. Nur so lassen sich die für diese Arbeit spezifischeren Fragestellungen von unerwartetem Verhalten untersuchen, die bisher bei einer allgemeinen Analyse der Praxistauglichkeit in der Literatur lediglich als Seiteneffekte beschrieben oder aus erfolgskosmetischen Gründen unterschlagen wurden.

Für die angesprochene einfache Realisierbarkeit spielt vor allem die Tatsache eine Rolle, dass die notwendige Hardwareinfrastruktur im Gegensatz zu Beispielen aus dem Mobilfunk- oder Automobilbereich leicht zu beschaffen und im Heimbereich auch vollständig und über einen längeren Zeitraum hinweg zu kontrollieren ist. Andere Szenarien wie beispielsweise ([21], [22] etc.) lassen sich ohne erheblichen Aufwand nur eingeschränkt und unter Laborbedingungen testen. Des Weiteren existieren dazu ebenso aus diesem Grund auch keine unabhängigen Publikationen über aufgetretene Probleme und kritische Punkte, die zum Vergleich herangezogen werden könnten.

2.1.1 Versuchsaufbau

Aus Gründen der Übersichtlichkeit und Realisierbarkeit ist das untersuchte Umgebungsmodell des Szenarios kein ganzes Haus, sondern lediglich ein zwei Zimmer Appartement. Es besteht aus einem Wohnzimmer, einem Schlafzimmer, weiterhin Küche und Bad. Alle Zimmer sind um einen zentralen Flur angeordnet, zum Wohnzimmer gehört ein Balkon (Abb. 7).

An elektrischen Geräten befinden sich über den gesamten Wohnraum verteilt eine Reihe von über ein Powerline Netzwerk steuerbaren elektrischen Verbra-

uchern (Kühlschrank, Mikrowelle, Waschmaschine, etc.) und diverse Lampen sowie zwei Servercomputer, die innerhalb eines 100Mbit/s Netzwerks eine größere Menge an Rechenleistung und Speicherkapazität zur Verfügung stellen.

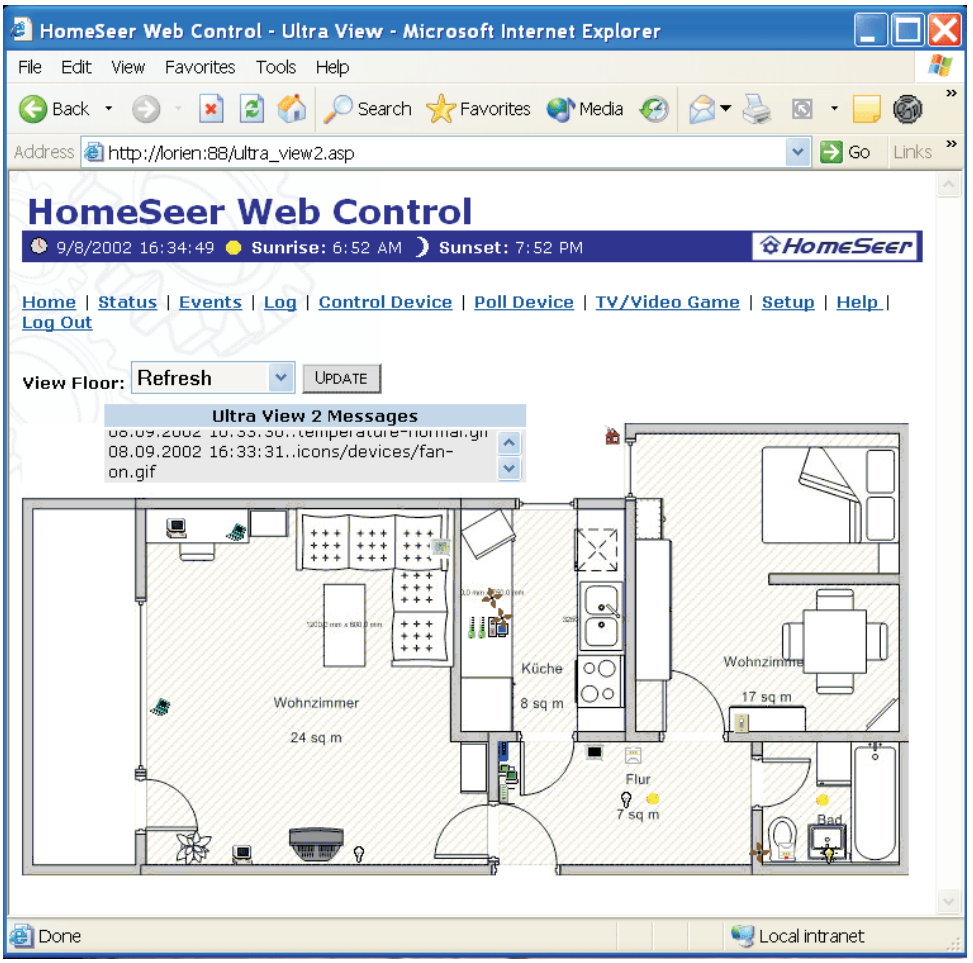


Abb. 7: Realisierter Teil des Umgebungsmodell im Szenario der Fallstudie

Zur direkten Interaktion mit den zahlreichen Anwendungsfunktionen stehen den Bewohnern in jedem Raum ein oder mehrere Konsolen zu Verfügung. Konsolen sind spezielle Geräte oder multifunktionale Bedienelemente, die nicht ausschließlich zur Steuerung eines bestimmten Gerätes oder einer Funktion dienen. Sie können für sich als eigene Geräte existieren, oder integriert in andere Geräte sein. Kennzeichnend sind universell einsetzbare Eingabeelemente (Funktionstasten, Touchscreens, Spracheingabe) und optionale Ausgabemöglichkeiten (Ton, Bild etc.). In jedem Raum befindet sich mindestens eine dieser Konsolen, im Flur zum Beispiel als eine Art von digitalem Bilderrahmen.

Zusätzlich zu den mehr oder weniger sichtbaren Geräten existieren zahlreiche kleine Sensoren. Sensoren sind sehr spezielle Geräte, die normalerweise nicht direkt wahrgenommen werden und mit denen nicht direkt interagiert wird (sonst können sie als Geräte betrachtet werden, z.B. Feuermelder, Thermometer etc.). Die Räume enthalten verschiedene Sensoren, etwa für Schalldruck, Luftfeuchtigkeit, Helligkeit, Temperatur, Bewegungsmelder, Öffnungssensoren an Türen und Fenstern sowie Lichtschranken an den Durchgängen.

2.1.2 Endbenutzer Szenario

Computerunterstützung in Alltagssituationen als typisches ubiquitäres Szenario

Das als Grundlage der Fallstudie betrachtete fiktive Szenario selbst besteht aus einer Beschreibung des typischen Tagesablaufes aus Sicht der Bewohner. Dieser beginnt um 5:30 Uhr. Während die Bewohner A und B noch schlafen, schließt sich das Fenster im Schlafzimmer selbsttätig. Gleichzeitig beginnt die Heizung zu arbeiten und Schlafzimmer, Küche, und Bad auf 22° zu erwärmen. Um 5:45 Uhr beginnt ein mehrstufiges Weckprogramm. Die Helligkeit im Raum wird stetig erhöht, dazu spielt auf Wunsch sanfte Musik. In der Küche wird die Kaffeemaschine aktiviert. Später öffnet sich die Jalousie im Schlafzimmer. In der Küche wird der Toaster gestartet und ein lautes Wecksignal ertönt. Auf den verschiedenen Terminals sind wichtige Schlagzeilen und Wetterberichte der beiden Orte zu sehen, an denen sich A und B am betreffenden Tage aufhalten werden.

Um 6:40 Uhr verlässt Bewohner A das Haus. Bewohner B ist fünf Minuten später bereit zu gehen, erhält jedoch die Meldung angezeigt, dass der nächste Nahverkehrszug 10 Minuten Verspätung hat. Bewohner B verlässt das Appartement folglich um 6:50 Uhr. Hinter ihm wird die Türe verschlossen. Das vergessene Licht im Bad wird gelöscht, die Wasserventile verriegelt, die Alarmanlage aktiviert. Der Hauptserver in der Küche wird deaktiviert, da keine aufwändige Spracherkennung mehr notwendig ist, ebenso wird die Warmhalteplatte der Kaffeemaschine deaktiviert. Die Temperatur wird in allen Räumen auf 18° gesenkt, im Schlafzimmer gelüftet.

In der Abwesenheit werden zudem alle Anrufe je nach Wichtigkeit direkt auf das Smartphone von Bewohner A oder auf einen Anrufbeantworter umgeleitet und von dort per Email als Sprachnachricht in das Büro von Bewohner B weitergeleitet.

Um 17:00 Uhr erhält Bewohner B eine SMS mit einer Erinnerung auf sein Mobiltelefon, den wöchentlichen Einkauf durchzuführen. Daran schließt sich eine automatisch erstellte Liste mit dem Wochenbedarf an Nahrungsmitteln an. Zu Hause angekommen wird die Eingangstüre automatisch bei An-

näherung entriegelt, das Licht im Flur schaltet sich ein, die Temperatur in allen Räumen außer dem Schlafzimmer wird erhöht. Das System teilt mit, dass Bewohner A wahrscheinlich gegen 19.00 eintreffen wird und aktiviert von sich aus eine Applikation für die Registrierung der neuen Einkäufe. Danach startet sanfte Musik in gedämpfter Lautstärke, während B die Einkäufe verstaut.

Das vollständige Szenario kann im Anhang (A2) nachgelesen werden. Für ein generelles Verständnis der Fallstudie und der zugehörigen Prototypen ist der betrachtete Ausschnitt jedoch völlig ausreichend, da es sich lediglich um ein fiktives Szenario handelt, das in der Fallstudie selbst nur partiell zu realisieren war. So musste beispielsweise aus Kosten- oder versicherungsrechtlichen Gründen auf den Einbau verschiedener Geräte und Funktionen (Eingangstüröffner, Hauptwasserabsperrung etc.) verzichtet werden. Die technisch noch nicht zu realisierenden Teile des Ausgangsszenarios dienen daher lediglich als Grundlage für Gedankenexperimente, Anforderungsanalysen und als Ausblick auf zukünftig mögliche Leistungen ubiquitärer Anwendungen.

2.2 Übersicht der Funktionalität

Primäre Funktionen

Auch wenn das gerade innerhalb des Szenarios betrachtete Beispielsystem oberflächlich als eine homogene Anwendung wahrgenommen werden kann, beschreibt das Szenario genau genommen eine Menge von Einzelapplikationen, die für sich gesehen bereits einen sinnvollen Zweck erfüllen:

- Eine Steuerung für elektrische Verbraucher (Geräte und Lampen)
- Vier Informationssysteme zur Anzeige von jeweils Gerätezuständen, Wetterdaten (Abb. 8) und Vorhersagen, Nachrichten und Zugverspätungen.
- Eine Multimediasteuerung zur Wiedergabe von Musik in mehreren Räumen
- Ein Telefonsystem zur Aufzeichnung und Weiterleitung von Anrufen.
- Ein Verwaltungssystem für die Registratur der Ein- und Ausgänge von Nahrungsmitteln (siehe Abb. 9).
- Eine Kochrezeptverwaltung mit Planungssystem.
- Ein digitaler Bilderrahmen, der auf den vier Terminals Fotos aus verschiedenen Quellen anzeigt, sofern keine andere Anwendung das Terminal beansprucht.

Infrastruktur-Funktionen

Hinzu kommen eine Reihe von Infrastrukturdiensten, die von einigen dieser Anwendungen gemeinsam genutzt werden:

- Ein Sprachdialogsystem, das in der Lage ist, Textausgaben in Sprache zu verwandeln und umgekehrt gesprochene Sätze in Textkommandos umzusetzen.
- Ein Webserver zur Anzeige verschiedenster Informationen auf unterschiedlichen Terminals.
- Ein Nachrichtensystem, über das Emails und SMS-Nachrichten verschickt werden können.

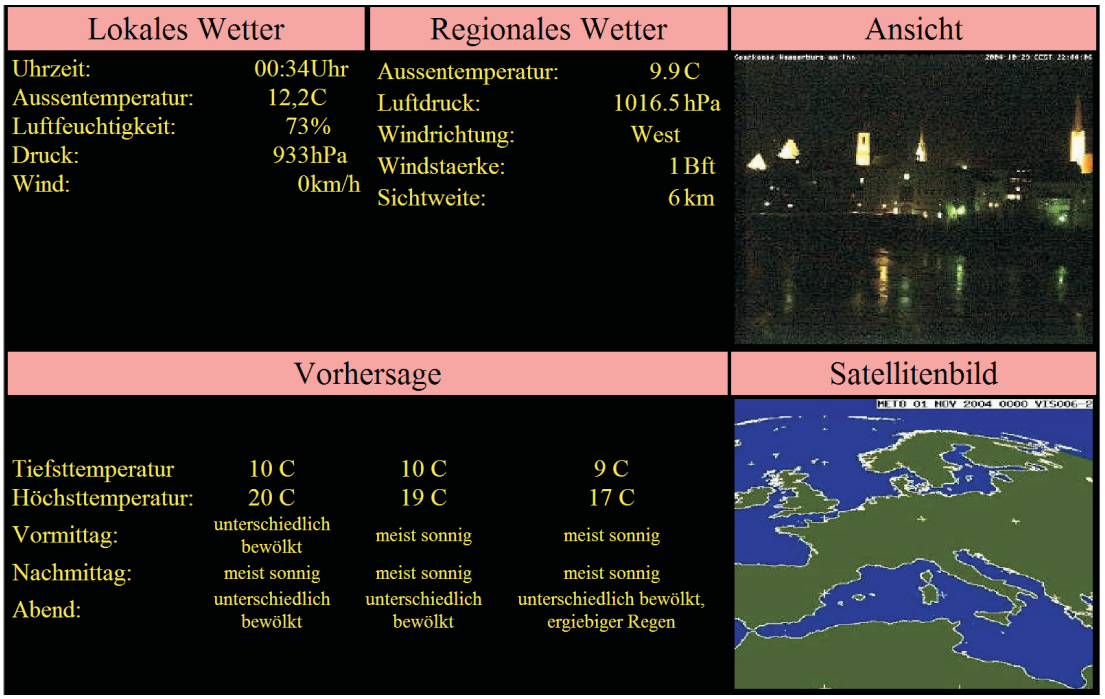


Abb. 8: ubiquitäres Wetterinformationssystem

In nichtubiquitären Systemen würden nun möglicherweise sowohl die Primär- wie auch die Infrastrukturfunktionen ständig verfügbar gehalten werden. Ausgenommen davon wären lediglich die Teile, die in einer konkreten Installationsinstanz überhaupt nicht benötigt würden und über entsprechende manuelle Konfigurationsmöglichkeiten bei der Erstinstallation deaktiviert werden können.

Aber selbst eine solche Einschränkung auf tatsächlich benötigte Funktionalitäten macht gerade im vorgestellten Szenario immer noch keinen vollständigen Sinn. Der notwendige Ressourcenverbrauch der restlichen bereitgehaltenen Funktionen in Form von Geräteabnutzung und elektrischem Verbrauch stünde nämlich in keinem Verhältnis zu der Anzahl der tatsächlichen Nutzungen über einen Tag verteilt (besonders Nachts). So verursacht beispielsweise

schon ein einziger PC im Dauerbetrieb Energiekosten von ca. 150€ pro Jahr, obwohl seine Leistungsfähigkeit nur die wenigste Zeit davon wirklich ausgenutzt würde. Zudem werden nicht immer alle Funktionen gleichzeitig eingesetzt. Zum Beispiel macht die Spracherkennung bei gleichzeitiger Wiedergabe von Musik oder anderweitig erhöhter Umgebungslautstärke mit den bisher zur Verfügung stehenden Erkennungstechnologien praktisch keinen Sinn.

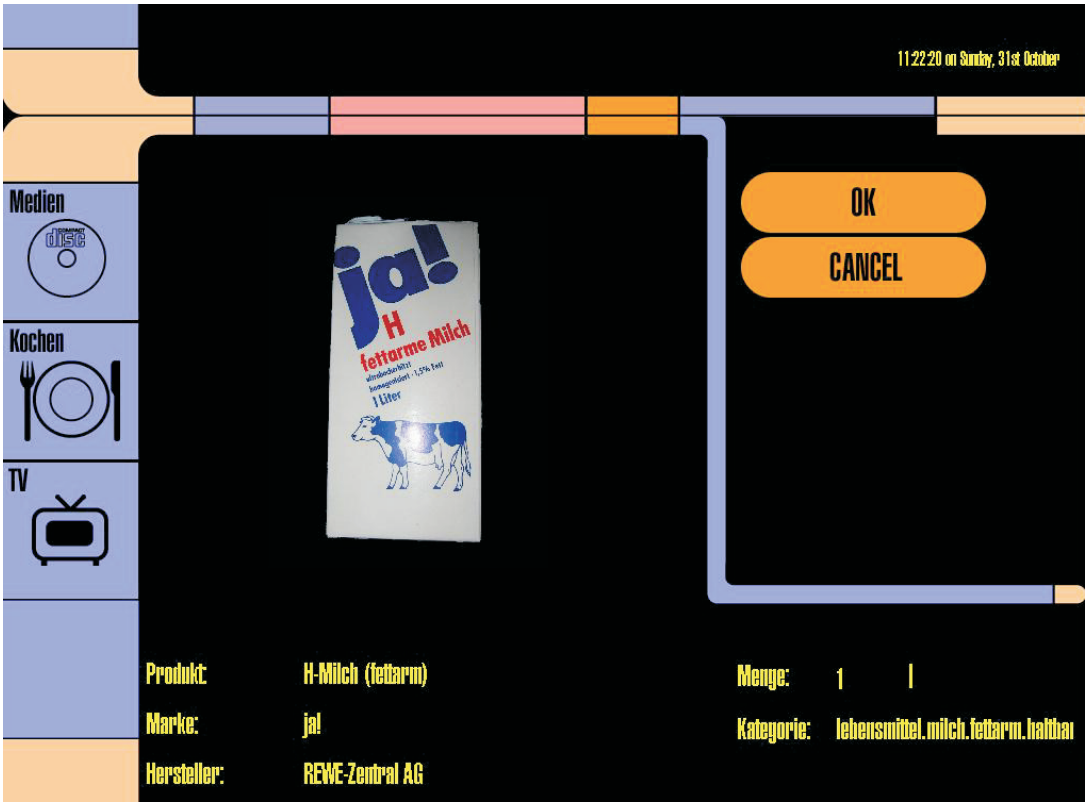


Abb. 9: Lebensmittellogistik-Funktion des Prototypen

In nichtubiquitären Anwendungen müsste folglich der Benutzer selbst eine ständige und manuelle Anpassung des Systems an die jeweiligen Umgebungsbedingungen vornehmen. Die Nutzung einer der oben beschriebenen Basisapplikationen sähe also in den meisten Fällen so aus:

- Manuelle Aktivierung der Geräte und ggf. Aktivierung der gerade benötigten Basisdienste (z.B. Anschalten des Hauptservers beim Betreten der Wohnung)
- Aktivierung des gewünschten Zugangsgerätes und Aufruf der Applikation.
- Eingabe von sich häufiger ändernden Personalisierungsinformationen (z.B. Ort, für den die Wettervorhersage gelten soll).
- Nutzung der Anwendung.

- Beendigung der Anwendung und Ausschalten eventuell nicht mehr verwendeter Zugangsgeräte und Server.

Dies entspricht in etwa dem gegenwärtig gängigen Nutzungsformen von Computern für Steuerungs- und Informationsaufgaben im Privatbereich. Ein solches Nutzungsschema würde aber dazu führen, dass ein Großteil der zur Verfügung stehenden Funktionalität in vielen Situationen des Bewohners nicht oder nicht komfortabel genutzt werden kann. So würde im Beispiel des Szenarios Bewohner B wohl kaum kurz vor dem Verlassen der Wohnung seinen Arbeitscomputer anschalten, um im Internet die Informationen über die Zugverspätung abzurufen. In anderen Fällen würde die Nutzung zeitaufwändig sequenzialisiert. Im Beispiel des Szenarios würde Bewohner B zum Beispiel zuerst im Internet nach dem Wetterbericht sehen und sich dann dementsprechend ankleiden. Das ist auch der Grund, warum die meisten Leute nach wie vor lieber den Wetterbericht im Radio hören. Allerdings hat dieser dort den Nachteil, dass er nur sehr allgemein für eine bestimmte Region zur Verfügung steht und nicht persönlich auf einen bestimmten Ort zugeschnitten werden kann.

Ubiquitous Computing löst nun diesen Nutzbarkeitskonflikt, indem die Anpassung an die jeweilige Situation automatisch und ohne direkte und bewusste Interaktion mit dem Benutzer erfolgt. Das bedeutet, es wird zunächst erkannt, welche Geräte und Softwarekomponenten überhaupt zur Verfügung stehen. Falls notwendig, können diese Komponenten auch automatisch aktiviert werden. Stehen mehrere Möglichkeiten zur Verfügung, kann darüber hinaus eine Realisierung ausgewählt werden, welche in einer bestimmten Situation am besten nutzbar ist. Im obigen Beispiel könnte also der personalisierte Wetterbericht auch vorgelesen werden und wäre damit einem einfachen Radioprogramm bei gleichem Nutzungskomfort überlegen. Da ubiquitäre Anwendungen zudem eine der einzelnen Anwendung übergeordnete Umgebungssituation mit einbeziehen können, entfiele auch die Eingabe der Personalisierungsinformation des Wetterberichtes, sofern diese bereits einmal an anderer Stelle, etwa in einem Kalender in Form des Ortes eines Termins eingetragen wurde.

Dieser übergeordnete Datenaustausch zwischen einzelnen Teilsystemen, die bestimmte Informationen in ihre Umgebung exportieren und damit eine Gesamtsituation erkennbar machen, erlaubt darüber hinaus, die ursprünglich isolierten Grundfunktionen zu ubiquitär nutzbaren Anwendungen zu kombinieren und deren Ausführung zum Teil zu automatisieren und damit auch zu parallelisieren. So wird im Szenario etwa die Verspätungsinformation automatisch abgerufen und auf einem Terminal in der Nähe des Bewohners oder durch die Sprachausgabe wiedergegeben. Beim Verlassen der Wohnung wer-

den dann alle technischen Ressourcen, die dafür notwendig waren automatisch deaktiviert. Durch geschickte Kombination der Grundfunktionen mithilfe von Adaption und Datenaustausch über die Situation lassen sich auf diese Weise sogar auch neue sekundäre Funktionen erzeugen:

Sekundäre Funktionen durch Anpassung an die Umgebungssituation

- Die Einkaufsliste entsteht durch die Kombination der Ausgaben der Inventarverwaltung der Nahrungsmittel über den durchschnittlichen Wochenverbrauch mit den Zutatenlisten der Rezeptplanung und dem Versenden über das Nachrichtensystem.
- Das automatische Ausschalten von Verbrauchern beim Verlassen der Wohnung entsteht durch eine Kombination von Informationen über die Gerätezustände (der Bewegungsmelder) mit der Applikation für die Gerätesteuerung.

Die durch die Infrastrukturfunktionalität der Anpassung an den Nutzungskontext gebildeten Sekundärfunktionen entstehen dabei lediglich temporär durch gewollte *Feature Interaction* [67]. Dies bedeutet, sie sind nur dann verfügbar, wenn auch die Grundfunktionen, aus denen sie sich zusammensetzen, ebenfalls verfügbar sind. Wäre beispielsweise das Nachrichtensystem außer Funktion, etwa weil der Telefonanschluss gestört ist, gäbe es auch keine Möglichkeit, die Einkaufsliste zu verschicken.

2.3 Aufbau und Realisierung des Prototyps I

Verwendung bestehender Komponenten und Architekturen

Eine der Hauptanforderungen für die Umsetzung des ersten Prototyp war, dass sie sich im Wesentlichen auf fertige Komponenten abstützen sollte. Damit wird ein neutraler Bezugspunkt für eine spätere Validierung des zweiten Prototyp erreicht. Eine Übersicht über die zurzeit verfügbaren Anwendungen zum Thema Heimautomation findet sich zum Beispiel in [52]. Technisch sind die einzigen in Frage kommenden Software-Pakete für die Nutzung von Computern im Heimbereich jenseits von PC oder speziellen Hobby-Anwendungen so genannte X10 [55] basierte Haussteuerungssysteme, mit denen verschiedene elektrische Geräte gesteuert werden können. Die X10 Hardware-Schalter kommunizieren über das Stromnetz (Powerline) oder eine Funkverbindung. Dadurch entfallen aufwändige Installationsarbeiten für Steuerleitungen. Weiterhin ist ein Steuerinterface erhältlich, das sich an einen herkömmlichen PC anschließen lässt (Abb. 10).

Als Ausgangsbasis für den Prototyp wurde schließlich das Programmpaket HomeSeer [53] ausgewählt. Entscheidend für die Auswahl waren günstiger Preis, gute Erweiterbarkeit und eine geringe Einarbeitungszeit, da das Pro-

gramm lediglich wichtige Basisfunktionen, wie das Auslesen und Verwalten der Zustände der verschiedenen zu steuernden Geräte in einer zentralen Datenbank, das Verschicken von Steuerkommandos und eine einfache regelbasierte Ereignisverwaltung beherrscht. Darüber hinaus gehende Funktionen können selbst in Form von Skripten nachgerüstet werden. Die einzige Open Source Alternative [54] bot unter der Voraussetzung, keine Änderungen am Sourcecode vorzunehmen, einen zu geringen Umfang an Basisfunktionalität. Alle weiteren in [52] beschriebenen Programmpakete waren entweder zu stark für einen bestimmten Zweck spezialisiert und dann mit zu vielen Sonderfunktionen überfrachtet oder nicht flexibel und erweiterbar genug, um für den geplanten Zweck geeignet zu sein.

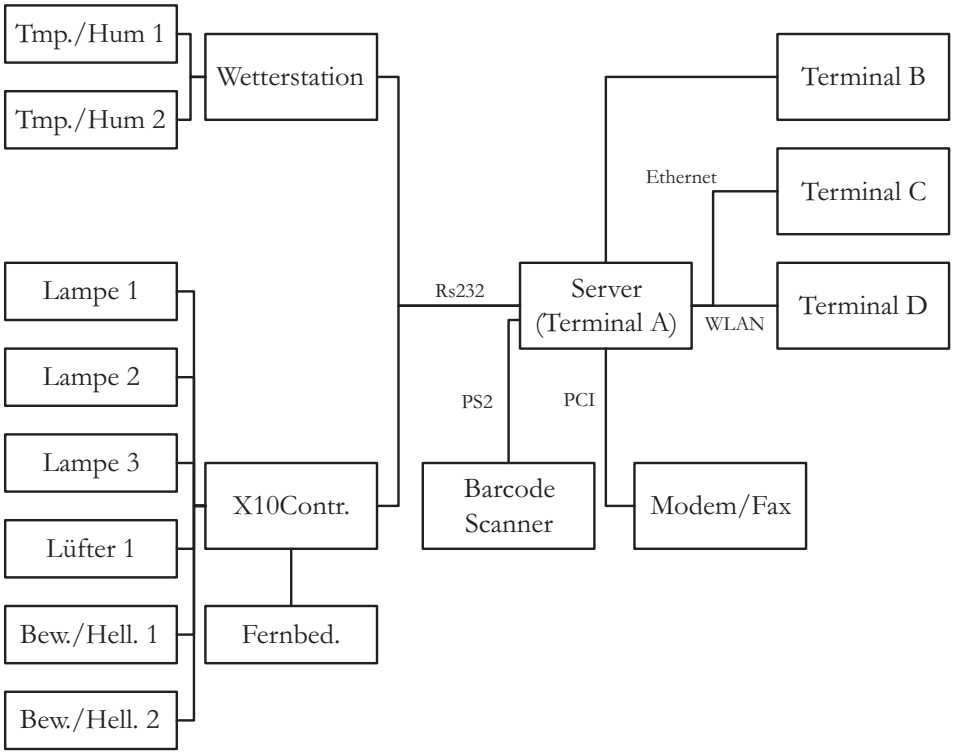


Abb. 10: Schematische Darstellung des ersten Prototyp

Zentrale Funktion des für den Prototyp I als Basis gewählten HomeSeer Programmpaketes ist eine einzige große Daten-Tabelle, in der normalerweise alle Geräte, deren Typ, Adresse, Zustände und weitere Informationen wie Datum und Zeit der letzten Änderung eingetragen sind. Über verschiedene Benutzerschnittstellen lassen sich die Werte in dieser Tabelle manuell verändern, was dann vom Basisprogramm in einen Schaltbefehl an die entsprechende Geräteadresse umgesetzt und über einen Gerätetreiber (X10 Control API) an das X10 Steuergerät übermittelt wird. In der anderen Richtung werden Änderungen der Gerätezustände, die vom X10 Steuergerät über das Stromnetz

empfangen werden, an die zentrale Datentabelle übermittelt. Zusätzliche Sensoren über die X10-Module hinaus, lassen sich als virtuelle Geräte in die Datenbank eintragen und über Skripte manipulieren. Damit ist bereits ein einfaches und nicht verteiltes Modell einer Umgebungssituation vorhanden. Bei jeder Änderung eines Wertes der Situation wird überprüft, ob dafür ein Ereignis definiert worden ist. Ein Ereignis besteht aus Werten bestimmter Elemente dieser Situation . Diese verschiedenen Werte können über boolsche Operatoren verknüpft werden und lösen so vordefinierte Aktionen aus. Dazu existieren im Grundprogramm verschiedene vordefinierte Aktionen, hauptsächlich zum Schalten weiterer Geräte über die Situation. Es können aber auch selbst vom Benutzer definierte Skripte gestartet werden, die Zugriff auf alle relevanten Teile des Systems haben. Damit ist auch bereits ein einfacher Adaptionsmechanismus realisiert. Die Architektur ähnelt den in vergleichbaren Arbeiten (z.B. [26]) beschriebenen wissenschaftlichen Referenzarchitekturen zum Thema Adaption und stellt damit gleichzeitig einen hinreichend akkuraten (Null-)Bezugspunkt für die Validierung des neuen wissenschaftlichen Beitrages dieser Arbeit dar.

Abb. 11 illustriert ein Beispiel der technischen Umsetzung einer Lichtsteuerung, die die auf die oben beschriebene Weise an die Umgebungssituation gekoppelt ist. Gesteuert wird sie abhängig davon, ob sich gerade jemand innerhalb eines dunklen Raumes befindet. Der Unterschied zu einer klassischen Lichtsteuerung besteht dabei in der Abstraktion von den konkreten Sensoren, welche die notwendigen Steuerinformationen (also die Situation) liefern. Dabei kann es sich wie im Beispiel um Infrarot-Bewegungsmelder handeln, genauso gut aber auch um Lichtschranken. Zusätzlich besteht die Möglichkeit, die Sensoren auch im laufenden Betrieb zu wechseln oder auf völlig neue, zum Zeitpunkt der Programmentwicklung noch unbekannte Sensoren zurückzugreifen.

Im Beispiel sendet der Bewegungsmelder bei Bewegung innerhalb des Raumes per Broadcast eine entsprechende Meldung über das Stromnetz. Diese Meldung wird vom X10 Steuergerät empfangen und an den Rechner und das HomeSeer Programm weitergeleitet. Die Information wird in der Geräte-Datenbank gespeichert und löst ein Ereignis aus. Dieses wiederum überprüft die Helligkeit im Raum und startet ein *Script*, das weitere Kontextinformationen überprüft (hier nur die Uhrzeit) und dann entscheidet, welche Aktion durchzuführen ist.

Diese Aufteilung der Adaption auf einen regelbasierten Eventmechanismus und Scriptprogramme gleichzeitig erscheint etwas inhomogen, liegt aber darin begründet, dass die Mächtigkeit der Regelauswertung im verwendeten Programmpaket stark eingeschränkt ist und deshalb nicht alle Überprüfun-

gen einfach erledigt werden können. Andererseits hat sich ein direktes Umleiten aller Ereignisse auf ein zentrales Script als zu langsam erwiesen, da die Initialisierung der Scripte eine gewisse Zeit in Anspruch nimmt und nur durchgeführt werden sollte, wenn auch wirklich eine Adaptionsaktion zu erwarten ist. Der Regelmechanismus führt deshalb eine Art Vorfilterung der teilweise sehr schnell aufeinander folgenden Geräteereignisse durch und erledigt einfache Schaltaufgaben. Für komplexere Aufgaben wird dann auf ein langsameres Script zurückgegriffen.

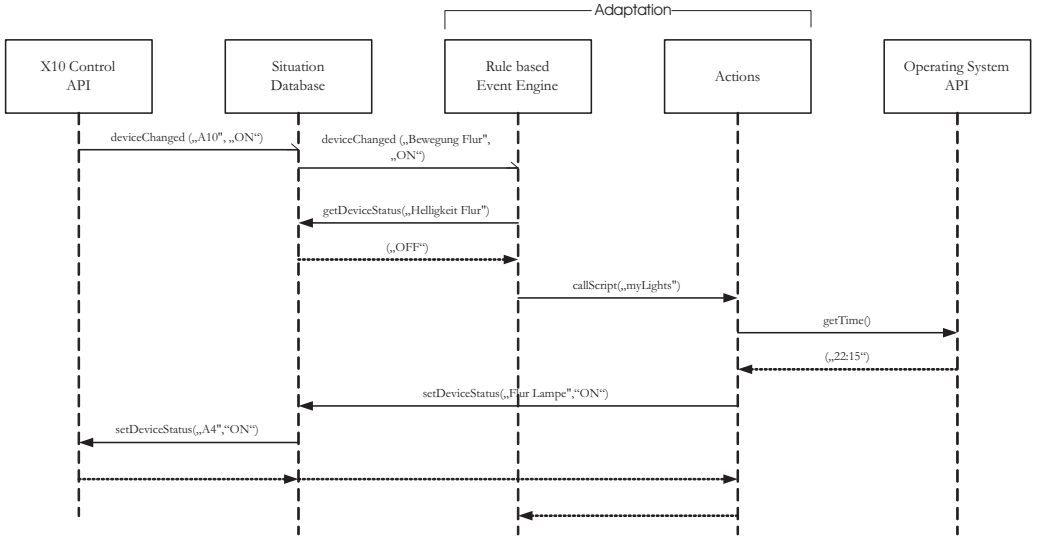


Abb. 11: Beispiel Lichtsteuerung

Die Realisierung der im Szenario beschriebenen situationsabhängigen Informationssysteme gestaltete sich ebenfalls sehr einfach. Es musste lediglich eine weitere Komponente hinzugefügt werden, welche die automatische Steuerung eines Webbrowsers auf einem Terminal erlaubt. Die jeweils darzustellenden Inhalte werden von der Adaptionslogik, d.h. in diesem Fall von einem Script bestimmt, welches die Terminal-Display Komponente kontextabhängig mit verschiedenen in Form von Webservices realisierten Informationsdiensten verbindet.

Abb. 12 zeigt ein Sequenzdiagramm für die Darstellung von Zugverspätungen auf dem Terminal, sofern sich Bewohner im Haus befinden und eine Zeit erreicht ist, zu der sich einer der Bewohner möglicherweise gerade vorbereitet, das Haus in Richtung Bahnhof zu verlassen. In diesem Fall wird durch die Regelauswertung zeitgesteuert alle 60 Sekunden ein Script (siehe Quelltext 1 unten) aufgerufen, das die notwendigen Kontextbedingungen überprüft und

bei positivem Ausgang dieser Prüfung eine bestimmte Aktion des Terminal Displays aufruft.

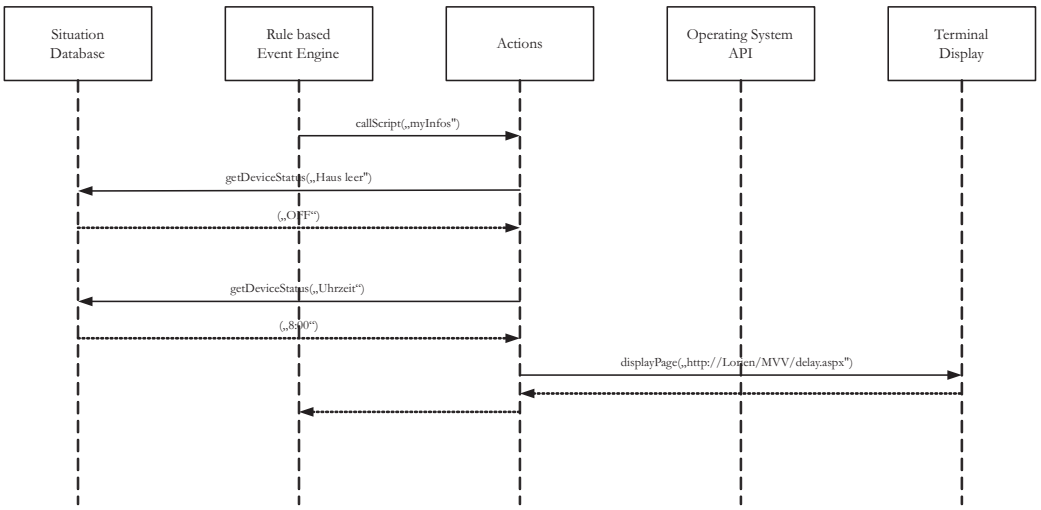


Abb. 12: Kontextabhängiges Informationssystem

```
sub gui()
  ...
  strInterface="http://lorien:88/slideshow.asp"
  // Default zeige Bilderrahmen, sonst vormittags Wetter und
  // kurz vor den Zugabfahrtszeiten die Verspätungsinformation

  if Hour(Now) < 11 then
    strInterface="http://lorien/Weather/ForeCastUI.aspx"
    if (Minute(Now) >= 0 and Minute(Now)<=7) or
       (Minute(Now) >= 20 and Minute(Now)<=27) or
       (Minute(Now) >= 40 and Minute(Now)<=47) then
         strInterface="http://bahnhofstafel.db-ris.de/bht/
         internet/tafel.html?AnAb=AB%26bhf=8000785"
    end if
  end if
  ...
  localCommand("show.html?"&strInterface)
  ...
end sub
```

Quelltext 1: Adaptionsregeln für Informationsdienste

2.4 SUB-Phänomene in der Praxis

Anhand des ersten im Rahmen der Fallstudie realisierten Prototyps wurden im Anschluss an dessen Inbetriebnahme über einen längeren Zeitraum hinweg sämtliche feststellebaren SUB Ereignisse registriert. Dies geschah im

Hinblick auf die Analyse der Zusammenhänge zwischen SUB-Phänomen und Frame-Problemen im nächsten Kapitel (3.2.1).

2.4.1 Erfassung von SUB

Wie schon in 1.2 beschrieben, handelt es sich bei *Spontaneous Unexpected Behavior* (SUB) um ein phänomenartiges Ereignis, das auch als eine Form des beiderseitigen Missverständnisses zwischen System und Benutzer interpretiert werden kann. Die in 1.2 genannten Symptome lassen sich aber nicht ohne weiteres von sporadisch auftretenden Programmier- oder Anforderungsfehlern einer Anwendung unterscheiden.

Erfassung nur unter realen Bedingungen in Langzeitstudien möglich

Über die Beobachtungsdauer von einem Jahr wurden daher alle Fälle registriert, in denen das Systemverhalten des Prototyps von den Nutzererwartungen abwich. Ebenfalls registriert wurde, ob das Fehlverhalten als kaum merklich, tolerierbar oder ernst (ärgerlich) eingestuft wurde.

Erst in der Auswertung wurden dann die Ursachen jedes einzelnen Fehlverhaltens analysiert. Im Gegensatz zu den reinen Symptomen kann nämlich anhand der Ursachen durchaus unterschieden werden, ob eine bestimmte Beobachtung der Klasse der Fehler (System war nicht korrekt) oder der Klasse der SUBs (System war korrekt aber in der Situation nicht geeignet, die Benutzererwartungen zu erfüllen) gehört. Die für diese Entscheidung verwendeten Kriterien werden im folgenden Unterabschnitt genauer beschrieben.

2.4.2 Kriterien

Als Ursachen für ein Verhalten, das nicht den Nutzererwartungen entspricht, kommen ganz prinzipiell die folgenden Möglichkeiten in Frage:

- Fehler in der Realisierung
- Fehler in den Anforderungen
- Nutzererwartungen, die von den der Anwendung zugrunde liegenden Anforderungen abweichen.

Nach der Begriffsdefinition für Fehler aus [71], ist SUB bei enger Auslegung der Definition kein Systemversagen (Failure, Fehler), da das beobachtbare Systemverhalten nicht von seiner Spezifikation abweicht, zumindest solange es sich dabei nicht um eine Anforderungsspezifikation mit von vornherein unerfüllbaren Forderungen handelt ("muss immer die Benutzerbedürfnisse erkennen" oder Vergleichbares). SUB-Phänomene gehören folglich in die dritte Ursachenkategorie und grenzen sich damit von anderem Fehlverhalten dadurch ab, dass das zugrunde liegende System nicht nur korrekt im Sinne seiner Spezifikation ist, sondern auch dessen Anforderungs- und technische

Spezifikation zum Zeitpunkt ihrer Erstellung korrekt bezüglich einer bestimmten geplanten Einsatzsituation (Umgebung, Verhalten des Benutzers, Bedürfnisse, Wirtschaftlichkeit etc.) war. Darüber hinaus grenzt noch das Spontanitätskriterium SUB von Fehlverhalten ab, das entsteht, wenn ein System von Anfang an außerhalb der vorgesehenen Einsatzumgebung eingesetzt wird. Zusammengefasst spricht man also von einem SUB-Phänomen, wenn die folgenden Kriterien allesamt erfüllt sind:

SUB sind keine Fehler

- Es existiert eine beobachtbare Abweichung zwischen Nutzererwartung und Systemverhalten.
- Das System verhält sich korrekt bezüglich seiner Spezifikation.
- Die Spezifikation erfüllt die Anforderungen.
- Die Anforderungen waren zum Zeitpunkt der Entwicklung für die geplante Einsatzumgebung korrekt.

Anders formuliert entsteht ein SUB einerseits immer dann, wenn der Benutzer die Fähigkeiten der Anwendung nicht kennt und dadurch unerfüllbare Erwartungen entwickelt. Eine andere Möglichkeit ist, dass sich die Bedürfnisse des Anwenders durch für das System nicht erkennbare äußere Einflüsse verändern oder während der Laufzeit bei der Entwicklung nicht vorhersehbare Einsatzsituationen entstehen, also das System von nicht mehr gültigen Annahmen (oder Anforderungen) ausgeht. Sehr drastische Beispiele hierfür wären etwa Währungsumstellungen (Euro), überlange Lebensdauer (Y2K) oder Timingprobleme durch zu schnelle Rechner. Kurzum alle Nichtfehler-Ereignisse, die normalerweise dazu führen würden, dass eine Anwendung durch eine neue Version ersetzt würde. Dies ist für ubiquitäre Anwendungen allerdings theoretisch unnötig, da sie sowieso immer neu zur Laufzeit aus verfügbaren Komponenten gebildet werden können. Die Ereignisse selbst können aber dennoch auftreten, auch in abgeschwächter Form, etwa als falsche Interpretation von Benutzerbedürfnissen aufgrund seltener Umweltphänomene, als unübliches Verhalten des Benutzers oder Ähnlichem.

Herausforderung ist Früherkennung von SUB bzw. Beseitigung durch den Benutzer

Die Herausforderung besteht vielmehr im Versuch, SUB-Phänomene im Vorfeld zu erkennen, zu vermeiden oder zumindest ohne Hilfe eines Entwicklers (und einer neuen Systemversion) durch den Benutzer beheben zu können.

2.4.3 Beispiele für SUB Ereignisse des Prototyp I

Trotz der stark eingrenzenden Kriterien konnten im Prototyp I wie für ubiquitäre Anwendungen vorhergesagt, eine große Zahl von SUB Ereignissen registriert werden.

Einschränkungen durch variable Umgebungsbedingungen

Ein guter Teil der registrierten SUB-Phänomene entfiel auf Ereignisse, die indirekt in Zusammenhang mit Fehlern von Komponenten außerhalb der betrachteten Anwendung standen. Obwohl es sich bei der eigentlichen Ursache um einen Fehler außerhalb des betrachteten Systems handelte, können die davon indirekt ausgelösten Abweichungen zwischen Nutzererwartung und tatsächlichem Systemverhalten dennoch in die Kategorie der SUB-Phänomene gehören, wenn es sich um ein ubiquitäres System handelt. Dann nämlich existieren in diesem Fall Anforderungen, dass das System bestimmte Funktionen unter variablen Umgebungsbedingungen so gut wie möglich erfüllt, also auch fehlertolerant arbeitet. In Kombination mit einer weiteren, in ubiquitären Anwendungen häufig auftretenden Anforderung nach der Nutzung lokaler Ressourcen entsteht eine Quelle für viele SUB Ereignisse, weil im Gegensatz zum Benutzer aus technischer Sicht die Unterscheidung zwischen einer fehlerhaften und einer nicht vorhandenen Ressource oft nur sehr schwierig oder überhaupt nicht zu lösen ist, wie das folgende Beispiel zeigt:

Kein Unterschied zwischen Ausfall und Nichtverfügbarkeit bei Umgebungsressourcen

> Die Anwendung des digitalen Bilderrahmens wurde unter der Anforderung konzipiert, mobile Geräte auf neue Bilder hin zu überprüfen, sobald diese im Netz verfügbar sind und diese bevorzugt anzuzeigen.
> Die Nutzererwartung zu dieser Anforderung bestand darin, dass nach der Rückkehr von einem Ausflug kurz nach Betreten des Wohnraumes die Bilder aus dem Foto-Handy über Bluetooth ausgelesen und auf den Terminals angezeigt würden. Durch einen Fehler des Bluetooth Stacks des Mobiltelefons nach einem Softwareupdate konnte jedoch in manchen Fällen keine erfolgreiche Verbindung zum Netzwerk hergestellt werden. War dies nach einem Ausflug der Fall, wurde ein SUB registriert, weil die Bilder nicht wie erwartet zu sehen waren.

In diesem Fall verhielt sich das System korrekt im Sinne seiner Spezifikation und seiner Anforderungen. Die Anforderungen waren ebenfalls korrekt. Die Forderung, Bilder aus Mobiltelefonen unabhängig von ihrer Verfügbarkeit im Netz zu extrahieren wäre dagegen unsinnig, da es keine objektive Möglichkeit gibt, zwischen einem nicht vorhandenem, fehlerhaftem oder bewusst ausgeschaltetem Mobiltelefon zu unterscheiden.

Subjektiv aus der Sicht des Benutzers existiert diese Möglichkeit dagegen schon. Dieser war der Meinung, dass man aus der Tatsache, dass zuvor im Schnitt täglich mindestens eine erfolgreiche Verbindung hatte hergestellt werden können, zumindest nach einiger Zeit eine Fehlermeldung zu erwarten gewesen wäre. Die Aufnahme einer solchen Anforderung wäre aber kontraproduktiv, da es bezogen auf die Menge aller möglichen Nutzer der Anwendung

viele andere Gründe für ein Ausbleiben einer Verbindung gibt, die dann wiederum ein SUB-Phänomen bezüglich der unnötigen Auslösung der Fehlermeldung verursacht hätten.

Korrekt arbeitende Toleranz gegenüber externen Fehlern zusammen mit der Nutzung von Ressourcen wechselnder Verfügbarkeit kann also SUB auslösen, wenn die externen Fehler nicht im Allgemeinen detektiert werden können.

Neben der Funktion des digitalen Bilderrahmens zeigten auch noch andere Module des Prototyps I eine Anfälligkeit gegenüber von externen Fehlern ausgelösten SUB-Phänomenen.

Einige der für die automatische Lichtsteuerung herangezogenen Bewegungsmelder basierten ebenfalls auf einer drahtlosen Übertragungstechnik und waren darüber hinaus batteriebetrieben. Wurden die Batterien schwach, oder war die Funkverbindung gestört, arbeiteten auch die Sensoren nur noch unzuverlässig.

Für die Lichtsteuerung bestand die Anforderung wiederum darin, in Räumen in denen Bewegung erkannt wurde das Licht zu aktivieren, sofern davon auszugehen war, dass die Helligkeit darin unzureichend war.

Wurden die Batterien eines Bewegungsmelders schwach, oder war die Verbindung unterbrochen, wurde ein SUB registriert, da die Beleuchtung ausblieb.

Gerade das letzte Beispiel zeigt sehr schön, dass die Unterscheidung zwischen Fehlerereignissen und SUB sehr stark davon abhängen kann, ob es sich bei der beobachteten Anwendung um eine ubiquitäre oder nichtubiquitäre Anwendung handelt. Eine nichtubiquitäre Anwendung wird für eine bestimmte Situation konzipiert. Zum Beispiel könnten die Anforderungen festschreiben, dass zum Betrieb der Anwendung in jedem Raum bestimmte und sogar redundante Sensoren erforderlich sind, deren Funktionsfähigkeit vom System überprüft werden kann. Das beschriebene Ereignis wäre dann ein Fehler (Programmierfehler, Installationsfehler etc.). In ubiquitären Anwendungen besteht dagegen oft die implizite Anforderung, eine bestimmte Funktion in möglichst vielen Situationen (z.B. unterschiedliche Art und Anzahl der Sensoren) so gut wie möglich zu realisieren. Das registrierte Ereignis ist damit eigentlich ein Feature, das für den Benutzer dagegen ein spontanes unerwartetes Verhalten (Licht bleibt aus, obwohl es drei Monate funktioniert hat) darstellt. Wie im Beispiel des Mobiltelefons ist eine Erkennung des Hardwarefehlers nicht objektiv möglich, etwa weil der betroffene Sensor im Gegen-

satz zu anderen für mehrere Tage seinen Zustand nicht ändert. Diese Art der Fehlererkennung würde nämlich potenziell in nicht generalisierbaren Ausnahmefällen wiederum zu SUB führen, beispielsweise wenn ein Sensor in einem Raum installiert wird, der nur selten betreten wird. Bliebe noch die Möglichkeit, Sensoren vorzuschreiben, die eine aktive Erkennung von Fehlfunktionen ermöglichen. Dies widerspräche aber den Ubiquitätsbedingungen. Die Anwendung würde dann nämlich in Situationen, in denen nur qualitativ schlechtere Sensoren zur Verfügung stehen, überhaupt nicht funktionieren, obwohl ein Betrieb (wenn auch mit größerer Unsicherheit) möglich wäre.

Neben den bisher beschriebenen Phänomenen traten in dieser Kategorie noch eine Reihe weiterer, durch externe Fehler verursachte SUB Ereignisse auf, die im Folgenden kurz zusammengefasst werden.

Durch den Ausfall eines Speicherbausteins im Hauptserver der Anwendung kam es zu einer deutlichen Reduzierung der Rechenleistung. In Folge dessen kam es zu einer Reaktionszeit bei der Aktivierung des Lichtes, die merkbar langsamer als gewohnt war. Das System verhielt sich korrekt und erfüllte nach wie vor seine Aufgabe. Für den Nutzer war die plötzlich auftretende permanente Verschlechterung der Qualität aber unerwartet und wurde daher als SUB registriert.

Die Informationsfunktion der Zugverspätung funktionierte korrekt, allerdings waren die von der Deutschen Bahn herausgegebenen Informationen zeitweise falsch. Dies führte dazu, dass einige Nutzer nicht wie erwartet ihren Zug erreichen konnten.

Die Wettervorhersagen eines bestimmten Anbieters erwiesen sich nach einiger Zeit plötzlich aus unbekannten Gründen als zu ungenau. Da die Informationen zuvor sehr zuverlässig waren, wurde ein SUB registriert.

Die drei letztgenannten Fälle von SUB sind gute Beispiele dafür, dass das Phänomen nicht ausschließlich auf ubiquitäre Anwendungen beschränkt ist. Vielmehr können auch längere und kontinuierliche Benutzung und ein damit einhergehender Gewöhnungseffekt dazu beitragen, dass eine Änderung der angebotenen Qualität als SUB registriert wird, obwohl ein System korrekt und innerhalb seiner in den Anforderungen beschriebenen Qualitätskriterien arbeitet. Ubiquitäre Anwendungen verstärken das Phänomen aber, indem sie eine größere Anzahl solcher kontinuierlich im Hintergrund zur Verfügung stehenden Funktionen integrieren. Für den Benutzer verschmelzen diese nämlich zu einer einzigen Anwendung, die möglicherweise unbenutzbar ist,

weil sie "ständig" irgendeine qualitativ minderwertige Funktion oder Information liefert.

Unvorhergesehene Nutzung/Kombination

Bei unbewußter oder indirekter Nutzung gibt es keine Bedienungsfehler

Ähnlich zur letzten Kategorie gibt es eine Klasse von SUB-Phänomenen, die von einem unvorhergesehenen Verhalten eines Benutzers herrühren. In nichtubiquitären Anwendungen kann ein solches Verhalten möglicherweise als Bedienungsfehler eingeordnet werden. Ubiquitäre Anwendungen halten sich jedoch oft im Hintergrund, so dass dem Benutzer nicht bewusst ist, dass eine bestimmte Tätigkeit als Eingabe gewertet wird und dass dafür ein bestimmtes "richtiges" Verhalten des Benutzers angenommen wird. Hinzu kommt, dass es für das System oft nicht entscheidbar ist, ob ein darauf bezogenes "unrichtiges" Verhalten des Benutzers aus Versehen oder mit voller Absicht geschah. Ein klassisches Trivialbeispiel für diese SUB-Kategorie sind automatische Rechtschreibkorrekturen, die vermeintlich falsche aber absichtliche Schreibweisen des Benutzers korrigieren. Im Prototyp konnten die folgenden SUB-Phänomene in diese Kategorie eingeordnet werden:

> Ein Benutzer verband die Telefonanlage mit einer Stromversorgung, die im Urlaubsmodus deaktiviert war. Für das System war nicht erkennbar, ob das mit Absicht geschah, weil das Gerät deaktiviert werden sollte, oder aus Versehen. Da es im gegebenen Fall ein Versehen war, wurde ein SUB registriert.

Oft stehen SUB Ereignisse dieser Kategorie in Zusammenhang mit schwierig oder gar nicht zu messenden Umgebungsbedingungen (siehe nächste Kategorie), die im Entwurf durch die Annahme eines konformen Nutzerverhaltens ersetzt werden.

> Eine Funktion des Prototyps bestand darin, bei Abwesenheit der Bewohner Telefonanrufe automatisch an das Mobiltelefon weiterzuleiten, sofern dieses aktiviert war. Allerdings bestand keine Möglichkeit, generell den Aufenthaltsort des Telefon zu ermitteln. Ein SUB wurde registriert, als der Benutzer das Telefon vergaß und gleichzeitig die Bluetoothverbindung deaktiviert hatte. Ersteres war dem Benutzer nach kurzer Zeit aufgefallen. Er hatte jedoch erwartet, dass vom System keine Gespräche auf ein als lokal erkanntes Gerät umgeleitet und statt dessen aufgezeichnet würden. Da die Bluetoothverbindung jedoch deaktivert, das Telefon aber erreichbar war, hatte das System angenommen, dass es vom Benutzer mit sich geführt wurde.

Falsche Interpretation von Messwerten/Messungenauigkeiten

Als eine weitere mögliche Quelle für SUB-Phänomene erwiesen sich bei der Beobachtung des Prototyps I Funktionen, die indirekt von schwer oder nicht direkt messbaren Umgebungsbedingungen abhingen. Deren Erfassung und Berücksichtigung gelingt prinzipiell nur mit einer eingeschränkten Genauigkeit und unter bestimmten teilweise impliziten Umgebungsannahmen. Letztere schließen Situationen aus, deren separate Erkennung und Behandlung, wenn überhaupt möglich, bezogen auf die sonstigen für eine Anwendung existierenden Anforderungen zu unwirtschaftlich sind, wie der folgende Fall zeigt.

Das folgende SUB-Beispiel bezieht sich auf eine Klimaregelung, die durch Kombination des Wetterinformationssystems mit der Gerätesteuerung von Fensteröffner und Heizungssteuerung realisiert wurde.

Die Anforderung bestand darin, an außerordentlich warmen Tagen während der Heizperiode bei einer gemessenen Außentemperatur von mehr als X° die Heizung auszuschalten und die Wohnung zu lüften.

Die Außentemperatur wurde dazu an einem Balkon redundant durch zwei im Permanentschatten angebrachte Sensoren gemessen. Trotzdem trat ein Fall ein, in dem durch eine ungünstige Kombination von Windstille, Sonnenstand, Strahlungsintensität und Bauweise des Balkons eine Situation entstand, in der die gemessene Temperatur um ca. 6° von der tatsächlichen Temperatur abwich. Durch das Öffnen der Fenster wurde es im Wohnbereich als Folge einem Nutzer zu kalt und daher ein SUB registriert.

Ein solches Systemverhalten ist kein Fehler, da in solchen und ähnlichen Fällen in der Regel bereits in der Anforderungsdefinition gewisse Unsicherheiten eingeplant und abgesichert werden. Im vorliegenden Fall etwa, dass bei einer Raumtemperatur von weniger als X° die Fenster geschlossen und die Heizung wieder aktiviert wird. Allerdings können die individuellen Toleranzschwellen eines Benutzers und die unterschiedliche Geschwindigkeit der Wahrnehmung bei Mensch und Maschine im Einzelfall abweichen, was zu dem beschriebenen SUB geführt hat.

Nichtvoraussetzung konkreter HW/SW Realisierungen führt zu Erkennungunsicherheiten.

Hinzu kommt im Fall ubiquitärer Anwendungen, dass diese in der Regel keine feste Einheit aus Hard- und Software bilden, also für verschiedene Situationen aus Empfindlichkeiten, Sensorqualitäten und Aufstellungsorte funktionieren sollen. Die Unsicherheit kann dadurch im Einzelfall stark variieren. Einfaches Ändern der Schaltschwelle ist dagegen auch keine Lösung für solche sporadischen Unsicherheiten, da dies einer Deaktivierung der Funktion gleichkäme (bei 15° Soll und einer Sicherheit von 6° ergäbe das eine durch-

schnittliche Schaltschwelle von 21°, was während der Heizperiode so gut wie nie erreicht werden würde).

Es ließe sich noch argumentieren, dass ein Fehler des Anwenders vorliegt, da das System möglicherweise außerhalb seiner Spezifikation betrieben wird. Dem widerspricht aber eine Reihe von Punkten. Erstens, dass diese Einschränkung der Anwendbarkeit nur implizit geschah, zweitens indirekt abgesichert war und drittens die kritischen Situationen des obigen Beispiels weder vom Nutzer noch vom System erkannt werden hätte können, bevor das SUB tatsächlich eingetreten ist.

Ein anderes Beispiel, das sich nicht auf Ungenauigkeiten sondern Unsicherheiten bei der Interpretation bezieht, ist das folgende SUB:

> Eine Sicherheitsanforderung des Prototyps bestand darin, dass das System in den Urlaubsmodus (Deaktivierung der meisten Funktionen) schaltet, auch wenn dieser nicht explizit aktiviert (also vergessen) wurde. In den Anforderungen wurde dazu als Bedingung genannt, dass die Bewegungssensoren darauf schließen lassen, dass während der Nacht niemand anwesend war (übernachtet hat). Die Bedingung war so gewählt, dass bis zur Erkennung in der Regel nicht mehr als ein Tag vergeht, also auch Wochenendurlaube sicher festgestellt werden können, andererseits abendliche Unternehmungen an einem Werktag nicht zu einer falschen Interpretation der Situation führen.
>
> Da es sich bei der Umgebungsbedingung Urlaub aber um einen schwer direkt messbaren Wert handelt, implizieren diese Bedingungen eine gewisse Unsicherheit. Zum Beispiel längeres Ausbleiben an Silvester oder zu besonderen Anlässen, was in einem Fall tatsächlich zu einem SUB aus Nutzersicht führte, da sich bei der Rückkehr das System im deaktivierten Zustand befand und gewohnte Funktionen (Licht, Status etc.) erst wieder nach dem Hochfahren aller Systeme zur Verfügung standen.

Gerade diese Steigerung von Messungenauigkeiten hin zu Interpretationsunsicherheiten illustriert sehr schön, dass es sich bei SUB nicht um Fehler handelt, obwohl aus Sicht des Benutzers ohne weitere Analyse normalerweise keine Möglichkeit besteht, alleine mithilfe der Symptome zwischen beiden Fällen zu unterscheiden. Das Ausbleiben der gewohnten Funktionen im letzten Beispiel hätte genauso gut von einem Programmierfehler oder Hardwaredefekt verursacht werden können oder einer unzureichend analysierten Anforderung.

Die Ursache war aber eine prinzipielle Messunsicherheit. Schließlich hätte die Anwendung in die Zukunft sehen oder die Gedanken des Nutzers beim Verlassen des Apartments lesen müssen, um eine hundertprozentig sichere Entscheidung zu treffen. Das gilt wohlgemerkt nur für den Fall einer ubiquitären Anwendung, andernfalls könnte man bereits das Vergessen der Urlaubsaktivierung durch den Benutzer als Fehlbedienung des Anwenders verbuchen, oder annehmen, dass zumindest immer eine Möglichkeit bestünde, sich beim Benutzer durch eine Rückfrage zu versichern.

Übertroffen wurden die beiden bisher in dieser Kategorie behandelten SUB Quellen aber noch von der bei oberflächlicher Betrachtung doch sehr trivialen Lichtsteuerung.

Diese sollte wie bereits in der letzten Kategorie beschrieben, eigentlich nur von der in einem Raum herrschenden Helligkeit abhängen. Zu Anfang war die Erkennung auf einen festen Schwellwert eingestellt. Diese Art der Adaption verursachte aber gleich eine Reihe von SUB-Phänomenen, die nicht durch ein einfaches Ändern der Schwellwerte beseitigt werden konnten. Bei längeren Beobachtungen der manuellen Bedienung stellte sich heraus, dass die Helligkeit, ab der ein Benutzer den Lichtschalter betätigte von seiner geplanten Aufenthaltsdauer, der geplanten Tätigkeit, der Tageszeit und der tatsächlichen Helligkeit (gegeben durch Wetter, Jahreszeit und Licht in anderen Räumen) sowie von der jeweiligen Position des Benutzers im Raum und der Person selbst abhing. Scheinbar spielten bei der Beurteilung der Notwendigkeit einer Beleuchtung verschiedene rationale (Energiesparen, Tätigkeit) und intuitive Überlegungen (schlechte Sicht) ebenso eine Rolle wie Gewöhnungseffekte. So scheint man abends generell eher bereit zu sein, eine Beleuchtung zu verwenden als tagsüber. Andererseits ist das Auge nach längeren Dunkelperioden empfindlicher und benötigt weniger Helligkeit als bei schnellem Wechsel zwischen Orten mit hoher und geringer Helligkeit. Nächtliche Bewegungen werden daher oft ohne Licht vollzogen, auch um andere Besucher nicht zu stören.

All diese Abhängigkeiten machten es genau genommen unmöglich, eine vollautomatische Lichtsteuerung zu erreichen, die in allen Situationen zufrieden stellend funktionierte. Eine Semiautomatik mit Lichtschaltern blieb der einzige Ausweg. Diese beschränkte die automatischen Schaltvorgänge auf die wenigen offensichtlichen Fälle und überließ dem Benutzer die Kontrolle in den schwierig zu entscheidenden Fällen.

Semiautomatikeffekte

Allerdings zeigte sich, dass Versuche, SUB Effekte durch manuelle Eingriffsmöglichkeiten zu kompensieren, wiederum neue SUB-Phänomene erzeugen können.

> Die im letzten Beispiel genannte Einführung einer Semiautomatik für die Lichtsteuerung, führte zu zwei neuen SUB-Phänomenen. Grund war, dass den Benutzern oft nicht klar wurde, wann die Beleuchtung manuell zu schalten war und wann die Automatik das Schalten übernahm. Dies führte dazu, dass im einen Fall der Benutzer auf die Automatik erfolglos wartete, während im anderen Fall Benutzer und Automatik nahezu gleichzeitig schalteten, was dazu führte, dass sich beide Kommandos im Ergebnis neutralisierten.

Funktionen die gleichzeitig automatisch und manuell aktiviert ausgelöst werden, können zu Bedienungsunsicherheiten führen.

Die beiden SUB-Phänomene dieser Kategorie sind ein besonders gutes Beispiel im Hinblick auf die später diskutierten Maßnahmen zur Verhinderung oder Beseitigung von SUB. Sie zeigen, dass es zum einen sehr wichtig ist, dem Benutzer ein klares Verständnis des Adaptionsverhaltens des Systems zu vermitteln. Dies kann auch bedeuten, dass Adaptionen bewusst vereinfacht werden müssen, beispielsweise durch das Beschränken der Lichtautomatik auf bestimmte Räume, in denen sie gut funktionieren kann. Ein Beispiel hierfür wären Kellerräume, in denen bei Betreten immer Licht benötigt wird.

Zum anderen zeigt das zweite Beispiel, dass sich SUB-Phänomene nicht immer einfach durch das Hinzufügen weiterer Informationen (im Beispiel der manuelle Schalter) beseitigen lassen. Statt dessen muss in solchen Fällen die komplette Interpretation einer Situation verändert werden. Im Beispiel also das Erstellen komplett neuer Regeln für die Automatik, die auch den Zeitpunkt des manuellen Kommandos berücksichtigen.

Überraschungen, Gewöhnungseffekte, und generelle Ausnahmen

Eine Reihe von weiteren Beispielen für SUB im Prototyp der Fallstudie entstand dadurch, dass im Ablauf der Anwendung eine bisher nicht aufgetretene neue Situation eintrat. Dem Benutzer wurde die neue Situation oft erst durch das Ergebnis der Adaption offenkundig. Dies äußerte sich für ihn in einem unerwarteten Systemverhalten. Dabei konnte es sich um Ausbleiben von gewohntem Verhalten aber auch um neues überraschendes Verhalten handeln.

Beispiele ließen sich vor allem in der Anfangsphase der Systemeinführung des Prototypen beobachten sowie beim Auftreten seltener Ereignisse. So wurde das erste (korrekte) Auftreten der zuvor beschriebenen Klimasteuerung (Öffnen der Fenster an warmen Wintertagen) als SUB registriert, weil sich bis dahin die Fenster nur zu festgelegten Zeiten geöffnet hatten.

Genauso fühlten einge Besucher und Bewohner zu Anfang ein gewisses Unwohlsein, das durch die scheinbare Intelligenz hinter den verschiedenen automatischen Aktionen verursacht wurde, besonders wenn diese physikalische Effekte wie Licht, Wärme oder Geräusche hervorriefen.

Manchmal werden aus wirtschaftlichen Gründen auch bestimmte als selten oder unwahrscheinlich erachtete Situationen bei der Entwicklung einer ubiquitären Anwendung von vornherein bewusst oder unbewusst ausgeschlossen. Ist dem Benutzer diese Einschränkung nicht bekannt, können aufgrund von Gewöhnungseffekten SUBs registriert werden. Dies ist ironischerweise besonders dann der Fall, wenn die Anwendung über einen längeren Zeitraum hinweg alle Bedürfnisse richtig erkannt hat, während bei schlecht funktionierenden Systemen die Wahrnehmungsschwelle für SUB Ereignisse sinken kann.

Ubiquitäre Systeme können leicht vom Benutzer unbemerkt in unbekannte Situationen.

Ein SUB wurde registriert, als vorübergehend ein weiterer Bewohner mit leichtem Schlaf anwesend war. Dessen nächtliche Bewegungen erzeugten im Zusammenspiel mit der automatischen Lichtsteuerung und Glaseinsätzen in den Türen eine Störung der anderen schlafenden Bewohner. Bauliche Besonderheiten waren aber in der käuflich erworbenen Lichtsteuerung nicht berücksichtigt worden.

Weitere SUB im Zusammenhang mit der Lichtsteuerung wurden registriert, als Bewohner bewusste Dunkelheit in einem Raum herstellen wollten, um die Nachtsicht nicht zu verlieren, bei Dunkelheit durch ein Fenster blicken oder das Aussehen eines Gegenstandes bei Dunkelheit zu betrachten. Als Folge davon wurde die manuelle Steuermöglichkeit nachgerüstet, was aber dann wie in der letzten Kategorie beschrieben wieder zu anderen SUB-Phänomenen führte.

Der feuchtigkeitsabhängig gesteuerte Lüfter erzeugte ebenfalls ein SUB, weil seine Geräuschkulisse in manchen Fällen nicht mit den Bedürfnissen des Benutzers nach Ruhe harmonierte.

Auch außerhalb der Gerätesteuerung ließen sich SUB Ereignisse dieser Kategorie registrieren. So hätte einer der gerade abwesenden Benutzer in einer Ausnahmesituation gerne sein Mobiltelefon verliehen, hatte aber keine Möglichkeit, die automatische Umleitung der Telefonanrufe zeitweise zu unterbinden.

2.4.4 Zusammenfassung und Diskussion

Die gezeigten Beispiele zeigen gut, dass die Mehrzahl von SUB Ereignissen in ubiquitären Anwendungen dadurch entstehen, dass vom System der Eindruck erweckt wird, sich vollkommen an die Bedürfnisse des Benutzers anzupassen. In Wirklichkeit gelingt dies aber nicht in jeder Situation. Je komplexer und weit gefasster die Funktionen einer ubiquitären Anwendung aber sind, desto häufiger kommt es zu Ausnahmesituationen, in denen das System nicht wie gewünscht reagiert und infolgedessen zu einem SUB.

Es gibt aber auch noch eine andere Seite. Da ubiquitäre Anwendungen sehr oft unsichtbar für den Benutzer im Hintergrund agieren und ihr Verhalten dadurch und durch Adaption schwerer nachzuvollziehen ist, weiß der Benutzer folglich nicht, was er von einem System letztendlich erwarten kann. Die Beispiele aus 2.4.3 lassen sich demnach grob in drei Kategorien klassifizieren:

- Die Situation ist für den Benutzer unterscheidbar, nicht jedoch für das System (z.B. Unterscheidung zwischen Gerätedefekt und Abwesenheit, bzw. neue unbekannte Situationen).
- Die Situation ist für das System unterscheidbar, nicht jedoch für den Benutzer (Gewöhnungseffekte, Überraschungen etc.).
- Die Situation ist für Nutzer und System unterscheidbar, wird jedoch unterschiedlich interpretiert (falsche Erkennung der Bedürfnisse, z.B. bei der Lichtsteuerung, Schlussfolgerungen aus veralteten oder zu allgemeinen Annahmen).

Ziel des folgenden Kapitels ist es nun, diese Zusammenhänge technisch im Bezug auf die Besonderheiten ubiquitärer Systeme zu erklären und auf ein prinzipielles und in der KI-Forschung schon länger bekanntes Problem zurückzuführen.

KAPITEL 3 Das Frame-Problem

Welcher Zusammenhang besteht zwischen Ubiquität und SUB?
Warum treten SUB Ereignisse in ubiquitären Anwendungen scheinbar vermehrt auf?
Wie sieht ein geeigneter Lösungsansatz aus?

Das folgende Kapitel erklärt den Zusammenhang zwischen den in Kapitel 2 beobachteten und diskutierten SUB-Phänomenen und einer wesentlichen Eigenschaft ubiquitärer Systeme, der Kontextadaption. Kontextadaption bedeutet kurz gesagt die explizite Selbstanpassung eines Systems, beispielsweise bezüglich seines Verhaltens, seiner Funktion oder Struktur an unterschiedliche Situationen seiner Einsatzumgebung. Für so eine selbstständige Erkennung einer Situation müssen Softwaresysteme jedoch in Form von Modellen auf konkrete Umgebungsmerkmale zurückgreifen, welche eine Situation charakterisieren (Kontext).

Die Schwierigkeit der Auswahl dieser Merkmale ist in der KI-Forschung bereits seit längerem als Frame-Problem [66] bekannt. Der Begriff beschreibt eine generelle Schwierigkeit bei der Erstellung eines Modells einer sich ständig ändernden, an sich unvorhersagbaren und unendlich komplexen Wirklichkeit [72]. In Verbindung mit teilweise als prinzipiell eingestuften Unzulänglichkeiten ubiquitärer Anwendungen wurde bereits des Öfteren ein problematischer Zusammenhang zwischen Frame-Problem und der Verarbeitung von Kontext in ubiquitären Systemen vermutet [23][70][73].

Ziel dieses Kapitels ist es, die verschiedenen Argumente bezüglich inhärenter Defizite ubiquitärer Anwendungen aufgrund des Frame-Problems zu diskutieren und insbesondere einen Zusammenhang zwischen dem in 2.4 beobachteten SUB-Phänomenen und den Auswirkungen des Frame-Problems herzustellen. Darüber hinaus wird ein spezieller und im Laufe dieser Arbeit

weiter verfolgter Softwareengineering Ansatz skizziert, der dabei helfen soll, das Frame-Problem und die damit verbundenen prinzipiellen Unzulänglichkeiten in ubiquitären Anwendungen zu umgehen.

3.1 Ursprung und Bedeutung

Wie spezifiziert man Änderungen, ohne unendlich viele Nichtänderungen auszuzählen.

Ursprünglich wurde mit dem Begriff des *Frame-Problem* lediglich ein eng definiertes technisches Problem im Zusammenhang mit dem *Situation Calculus* [74] aus der logikbasierten KI bezeichnet. Dieses Problem bestand darin, wie es unter Verwendung mathematischer Logik möglich ist, eine Formel anzugeben, welche die Auswirkung einer Aktion beschreibt, ohne eine große Anzahl von weiteren Formeln angeben zu müssen, die beschreiben, was sich durch die Ausführung einer Aktion nicht ändert.

Zur Illustration der Problematik ist folgendes kleines Beispiel [32] geeignet: Eine *Situation* kann als Schnappschuss der Welt zu einem bestimmten Zeitpunkt verstanden werden. In diesem Zusammenhang kann der *Zustand On(BlockA,BlockB)* als eine Ansammlung von Situationen betrachtet werden, in denen BlockA auf Block B liegt. Ein Zustand ist damit ein *Fluent*, kann sich also mit der Zeit verändern. Das Prädikat *Holds* setzt Situationen und Zustände miteinander in Beziehung, beispielsweise *Holds(S77,On(BlockA,BlockB))*. *Aktionen* sind Funktionen auf Zuständen. Die Aktion *Puton(BlockA,BlockB)* bildet zum Beispiel die Situationen ab, in denen BlockA und BlockB *Clear* (nichts darauf) sind. *Result* ist eine Funktion, welche die Situation vor einer Aktion auf die danach abbildet.

Mit *Holds(S0,Clear(BlockA)); Holds(S0,Clear(BlockB)); Holds(Result(Puton(BlockA,BlockB),S0),On(BlockA,BlockB))* kann man daher ausdrücken, dass BlockA auf BlockB liegt, nachdem er auf ihn gelegt wurde.

Fügt man jedoch die Tatsache hinzu, dass BlockA am Anfang rot ist, also *Holds(S0,Red(BlockA))*, stellt sich die Frage, ob daraus zu folgern ist, dass BlockA immer noch rot ist, nachdem er auf BlockB gelegt wurde, also *Holds(Result(Puton(BlockA,BlockB),S0),Red(BlockA))*?.

Der Situation Calculus lässt darüber keinen Schluss zu, solange keine Axiome darüber existieren, welche Fluents sich bezüglich bestimmter Aktionen nicht verändern. Axiome der Art wie $Holds(s,Red(BlockA)) \Longrightarrow Holds(Result(Puton(BlockA,BlockB),s),Red(BlockA))$ heißen daher Frame-Axiome.

Natürlich bleibt in halbwegs komplexen Modellen eine riesige Anzahl von Dingen bei der Ausführung einer Aktion unverändert. Die Notwenigkeit, eine große Menge von Frame-Axiomen hinzuzufügen, lediglich um zu beschreiben, was sich bei einer bestimmten Aktion nicht verändert, wurde als das ursprüngliche Frame-Problem bezeichnet.

Kurz gesagt, beschrieb die erste Verwendung des Begriffes die Frage, wie sich auf effiziente Weise feststellen lässt, welche Teile einer veränderlichen Welt unverändert bleiben [74]. Die Bezeichnung "Frame-Problem" wurde dabei nach einer gängigen Technik (framing) für die Herstellung von Trickfilmen geprägt, in der die beweglichen Teile mit einem statischen Hintergrund (frame) zusammenmontiert werden.

Dass es sich bei dem Frame-Problem um ein wirkliches Problem handelt, hat gleich mehrere Gründe. Zum einen, weil es sehr aufwändig sein kann, alle für einen bestimmten Schluss notwendigen Frame-Axiome zu formalisieren, zu speichern und zu durchsuchen. Zum anderen, weil Frame-Axiome sehr oft falsch werden können [102], insbesondere wenn nebenläufige Aktionen zugelassen werden.

Wenn beispielsweise jemand BlockA in einer anderen Farbe lackiert, während gleichzeitig die *Puton* Aktion durchgeführt wird, hat am Ende nach der Durchführung der Aktion BlockA seine Farbe sehr wohl verändert. Das System mit den zuvor beschriebenen Frame-Axiomen geht folglich von falschen Annahmen aus und kann, bezogen auf die Wirklichkeit zu falschen Schlussfolgerungen gelangen.

Im Situation Calculus ist die Nebenläufigkeit von Aktionen daher ausgeschlossen. Prinzipiell bleibt das Problem aber in einer mehr generalisierten Betrachtungsweise bestehen. Angefangen mit dem sehr spezifischen Problem der KI hat der Begriff des Frame-Problems daher mittlerweile eine breitere Bedeutung im Bereich der Philosophie und Wissensrepräsentation erlangt. Manchmal spricht man auch von einer "Familie" von Frame-Problemen, die aus den folgenden Fragen [75] heraus entstehen:

1.) welche Dinge (Fakten etc.) ändern sich in einem Modell und welche nicht?

2.) was sind die notwendigen und hinreichenden Bedingungen für ein Ereignis?

3.) wie kann 1.) repräsentiert werden und wie können aus 1.) Schlussfolgerungen gezogen werden?

Zu besagter Familie von Problemen gehören beispielsweise das *Persistence Problem, Temporal Projection Problem, Inertia Problem, Qualification Problem, Ramification Problem, Extended Prediction Problem, Installation Problem, Planning Problem, Holism Problem, Relevance Problem*.

Andere Betrachtungsweisen legen nahe, dass es sich bei den verschiedenen Ausprägungen des Frame-Problems lediglich um Symptome eines wesentlich fundamentaleren Problems handelt. Je nach Partei innerhalb eines teils sehr heftigen wissenschaftlichen Diskurses werden hierfür entweder das Problem der vollständigen Beschreibung [75], das Induktionsproblem [76] oder das Symbol Grounding Problem [77] als heiße Kandidaten gehandelt. Die verschiedenen Betrachtungsweisen des Frame-Problems sind allerdings allesamt sehr ähnlich und unterscheiden sich lediglich im gewählten Erklärungsansatz und (teilweise philosophischen) Abgrenzungen zu anderen Problemfeldern.

3.1.1 Bedeutung des Frame-Problems für die Softwaretechnik

Darstellungen von Modellen haben keine intrinsische Bedeutung

Aus den bisherigen Betrachtungen wird schnell klar, dass das Frame-Problem besonders im Zusammenhang mit typischen Fragestellungen der KI und Robotik eine große Relevanz besitzt. Anwendungen aus beiden Feldern agieren in einer veränderlichen Umwelt und sind darauf angewiesen, darin Situationen zu erkennen, oder herbeizuführen. Im ersten Fall also abzuschätzen, welche Auswirkungen eine beobachtete Änderung auf möglicherweise nicht direkt beobachtbare Eigenschaften von Objekten hat.

> Beobachtet ein Roboter beispielsweise, dass eine Person den Raum durch eine Türe verlässt, sollte er aus dieser Information darauf schließen, dass diese Person sich möglicherweise im angrenzenden Raum aufhält, obwohl er diese Tatsache nicht direkt beobachten kann. Genauso gut könnte er aber aufgrund des Frame-Problems auch zu dem falschen Schluss gelangen, dass die betreffende Person aufgehört hat zu existieren, oder dass keine Informationen über den Aufenthaltsort vorliegen. Dies kann fatale Folgen haben, wenn der Roboter sich entschließt, die betreffende Türe zu blockieren und dies der einzige Ausgang des betreffenden Raumes war.

Der zweite Fall von typischen KI Problemen ist die Planung notwendiger Aktionen, um ein bestimmtes Ziel zu erreichen, also eine gewünschte Situation aktiv herbeizuführen.

Dies beginnt bei der Planung einfacher Bewegungsabläufe in einer durch bewegliche Gegenstände veränderlichen Umwelt und endet bei der Planung von Strategien zur Erreichung eines Ziels, das andere intelligente Entitäten aktiv zu verhindern suchen. Kann aufgrund des Frame Problems die Wirkung einer bestimmten Aktion nicht vorhergesehen werden, bedeutet dies einen Unsicherheitsfaktor für die erfolgreiche Erfüllung einer Aufgabe.

Ganz offensichtlich spielen diese beide Anwendungsfälle mit hoher Frame-Problem Relevanz auch im Ubiquitous Computing eine Rolle. Ubiquitäre Anwendungen agieren ebenfalls in einer veränderlichen Umgebung aus verschiedenen Verfügbarkeiten von Hard- und Software sowie sich ändernden Benutzerbedürfnissen. Letztere können zudem in der Regel nicht direkt beobachtet werden, sondern müssen aus indirekten Beobachtungen heraus gefolgert werden.

Aus der Tatsache, dass ein Benutzer einen dunklen Raum betritt, könnte die ubiquitäre Anwendung ableiten, dass das Bedürfnis nach Licht besteht. Aufgrund des Frame-Problems kann diese Schlussfolgerung aber falsch sein, weil beispielsweise die notwendigen Frame-Axiome fehlen, die berücksichtigen, ob die betreffende Person etwa blind ist, gerade ihr neues Nachtsichtgerät testet, oder schlicht nur andere nicht stören will.

Selbst wenn die Bedürfnisse richtig erkannt sind, müssen sie durch die ubiquitäre Anwendung auch noch in einer veränderlichen Systemumgebung realisiert werden. Dazu muss zunächst geplant werden, welche Funktionen für die Erfüllung eines Bedürfnisses angewendet werden sollen und wie diese Funktionen dann mit der verfügbaren Infrastruktur realisiert werden, respektive, wie die notwendige Infrastruktur überhaupt erst verfügbar gemacht werden kann.

Für manche ubiquitär zu nutzenden Funktionen müssen beispielsweise aufgrund der notwendigen Rechenleistung zuerst bestimmte Geräte aktiviert werden, bevor etwa eine Spracherkennung zur Verfügung steht.

Die Bedeutung des Frame-Problems für Ubiquitous Computing liegt folglich auf der Hand. Weniger offensichtlich jedoch ist, an welcher Stelle die Grenze zu ziehen ist. Ab wann ist ein System also "intelligent" und seine Umgebung "veränderlich"? Wie viel Intelligenz und Änderung ist notwendig, damit das Frame-Problem eine Rolle spielt, beziehungsweise gibt es Anwendungen, die niemals durch Frame-Probleme betroffen sind?

Zur Beantwortung dieser Fragen und auch im Hinblick auf eine Lösungsmöglichkeit muss dazu der genauere Zusammenhang zwischen Softwaresystemen im Allgemeinen und dem Frame-Problem genauer untersucht werden.

Am einfachsten erschließt sich dieser Zusammenhang anhand des Symbol Grounding Problems, einer zuvor bereits erwähnten speziellen Deutung des allgemeiner gefassten Frame-Problems. Dieses bezeichnet die Schwierigkeiten, die Wirklichkeit mit formalen Symbolen zu verknüpfen, die eine intrinsische also selbstständige Bedeutung besitzen (grounded). Alle üblicherweise verwendeten Formalismen einschließlich der menschlichen Sprachen, die mehr als sich selbst beschreiben, besitzen jedoch nur eine extrinsische Bedeutung (ungrounded). Die Bedeutung entsteht dort lediglich durch ihre systematische Interpretierbarkeit und Verwendung im menschlichen Bewusstsein. Daher lassen sich zum Beispiel auch unbekannte alte Sprachen ohne Bezüge zu bereits bekannten Sprachen oder Ereignissen nicht entschlüsseln.

Geht man nun von einer unendlich oder zumindest sehr komplexen Wirklichkeit aus, so gelingt deren Wahrnehmung ausschließlich über Vereinfachungen in Form von Modellen. Dies gilt für Menschen, insbesondere aber auch für Softwaresysteme.

Definition 5: *Modelle* in der Softwaretechnik

"Ein Modell ist eine idealisierte, vereinfachte [...] Darstellung eines Gegenstandes, Systems oder sonstigen Weltausschnitts mit dem Ziel, daran bestimmte Eigenschaften des Vorbilds besser studieren zu können." [15]

Aufgrund des oben erwähnten Symbol Grounding Problems kann jede Repräsentation eines solchen Modells lediglich in einer Form geschehen, die auf extrinsische Bedeutungen angewiesen ist. Anders wäre die angestrebte Vereinfachung eines Modells gegenüber der Wirklichkeit auch nicht zu erreichen. Jedes Modell basiert folglich auf einer unendlichen oder zumindest sehr großen Anzahl von impliziten Annahmen.

Ein Teil dieser Annahmen ist als unveränderlich (statisch) angenommen, zumindest wenn sie sich auf die Relevanz des modellierten Wirklichkeitsausschnittes beziehen oder die Grenzen markieren, innerhalb derer sich ein Modell verändern kann.

Die modellierte Wirklichkeit dagegen ist in der Regel als veränderlich (dynamisch) anzunehmen, zumindest wenn man davon ausgeht (und viele physikalische Phänomene deuten darauf hin), dass zufällige Ereignisse und die Unmöglichkeit, die Zukunft vorherzusagen tatsächliche Eigenschaften der

Wirklichkeit und nicht bloße Defizite in unseren eigenen Wahrnehmungsmodellen sind.

Die Folge davon ist, dass Annahmen, auf denen eine statische Annäherung einer dynamischen Wirklichkeit in Form von Modellen basiert, über einen gewissen Zeitraum hinweg betrachtet falsch werden können. Dieser Umstand war ja bereits im Zusammenhang mit dem Ursprung des Frame-Problems im Situation Calculus mit seinen explizit gemachten impliziten Annahmen in Form der Frame-Axiome dargelegt worden.

Da die Annahmen in Modellrepräsentationen aber aufgrund ihrer schieren Menge zum großen Teil auch nur implizit durch den Bezug auf extrinsische Bedeutungen (also ungrounded) vorhanden sind, können diese falsch gewordenen Annahmen auch bei überaus sorgfältiger Modellierung nicht in jedem Fall aus dem Modell heraus überprüft und erkannt werden. Modelle können infolge dieses "Framing" Problems "veralten", nicht mehr mit der Wirklichkeit übereinstimmen und dadurch bei ihrer Verwendung falsche Ergebnisse produzieren. Ein Beispiel für die Auswirkungen sind SUB-Phänomene die sich darauf zurückführen lassen, dass eine Situation zwar für den Benutzer, nicht jedoch für das System erkennbar und unterscheidbar ist (siehe 2.4.3f).

3.1.2 Zusammenfassung

Kurz gefasst ist das Frame-Problem eines der harten Probleme repräsentationsbasierter Künstlicher Intelligenz und weiter gefasst aller Modelle der Wirklichkeit.

Das Problem besteht in der Schwierigkeit der Festlegung, welche Aspekte der Wirklichkeit in einem ausreichend detaillierten Weltmodell berücksichtigt werden müssen und wie ein solches Weltmodell auf dem neusten Stand gehalten werden kann, wenn sich die Wirklichkeit verändert.

So gesehen betrifft das Frame-Problem jedes Softwaresystem, das nicht auf einer eng begrenzten Menge von vorher festgelegten Daten (sog. *Toy-Systems* [78]) operiert. Es kann dabei in zwei Formen auftreten. Zum einen können die den Modellen zugrunde liegenden Annahmen veralten, was zu falschen Ergebnissen führt. Zum anderen können falsche oder veraltete Modelle nicht in jedem Fall sofort als fehlerhaft erkannt werden.

3.2 Frame-Probleme der Kontextadaption

Der Umstand, dass Modelle und damit auch Softwaresysteme auch bei sorgfältiger Konstruktion nach einiger Zeit (spontan) veralten und falsche Ergeb-

nisse produzieren können, ist für sich genommen keine große Überraschung. Die gerade durchgeführte Betrachtung des Frame-Problems erklärt folglich, dass das Auftreten von spontanem Fehlverhalten in ursprünglich korrekten Softwaresystemen auf das Frame-Problem zurückzuführen ist. Weniger offensichtlich ist jedoch, warum das Altern eines Modells auch von außen nicht immer zuverlässig erkannt werden kann, bevor es zu einem Verhalten (behavior) kommt, das nicht nur spontan (spontaneous), sondern auch unerwartet (unexpected) ist und zudem im Fall der Softwaresysteme besonders im Ubiquitous Computing gehäuft auftritt.

3.2.1 Warum ist SUB eine Instanz des Frame-Problems?

Um den Zusammenhang auch zwischen der Unerwartbarkeit spontanen Fehlverhaltens und dem Frame-Problem herzustellen, greift man wiederum am einfachsten auf die Erklärung des Symbol Grounding zurück. Die Tatsache, dass eine Repräsentation eines Modells (oder jeder andere symbolbasierte Formalismus) keine intrinsische Bedeutung besitzt, bedeutet nämlich auch, dass sie mehrere verschiedene Interpretationen haben kann. Das macht man sich ganz leicht anhand natürlichsprachlicher Wörter klar, die in mehreren Sprachen aber auch in ein und derselben unterschiedliche Bedeutungen haben können.

Normalerweise stellt jedoch eine gewisse Systematik bei der Interpretation sicher, dass Symbole in einem bestimmten Anwendungsgebiet auf die richtige Weise interpretiert werden. Dieses Verfahren funktioniert erstaunlich gut, solange es sich bei Ausgangs- und Zielpunkt einer solchen Symbolik um Menschen handelt, welche die externen Bedeutungen kennen und der beschriebene Sachverhalt Bestandteil eines gemeinsamen Wahrnehmungsbereiches der Wirklichkeit ist.

Missverständnisse können jedoch (auch zwischen Menschen) immer dann auftreten, wenn einer der Beteiligten eine extrinsische Bedeutung nicht kennt, oder der beschriebene Sachverhalt nicht Teil eines gleichwertigen Wahrnehmungsbereiches ist. Insbesondere ist dies der Fall, wenn der zu beschreibende Sachverhalt das Wahrnehmungsmodell einer anderen Person darstellt. Dessen extrinsische Bedeutung des Modells an sich (nicht der dadurch modellierten Wirklichkeit), also die eigentliche Interpretation entzieht sich in der Regel der Wahrnehmung anderer Personen. Aus diesem Grunde ist es auch für Menschen schwer, die Bedürfnisse, Wünsche, Gefühle, Erwartungen oder Reaktionen eines anderen vorherzusehen. Natürlich können diese durch den anderen mitgeteilt werden, die eigentliche Interpretation, welche dazu geführt hat, bleibt aber verborgen, da zwischen den beiden Personen eine unterschiedliche Wahrnehmungsperspektive (der Eine kann seine eigenen Gedan-

UC benötigt für die Berechnung von Benutzerbedürfnissen ein Modell über die Situationsinterpretationen der Benutzer

ken "lesen", der Andere jedoch nicht) besteht. Auf Softwaresysteme im Ubiquitous Computing trifft das noch im Besonderen zu, da diese sich für die Erkennung von Benutzerbedürfnissen lediglich auf die zum Entwicklungszeitpunkt fixierten extrinsischen Bedeutungen ihres Konstrukteurs beziehen können, während die extrinsischen Bedeutungen des Benutzers sich durch zeitlich nachfolgende Ereignisse verändern können.

Vergleicht man beispielsweise die Preisauszeichnung eines deutschen Produktes ohne Währungsangabe (z.B. 10.-) heute und vor 20 Jahren, stellt man fest, dass sich die zugehörige extrinsische Bedeutung aufgrund eines bestimmten Ereignisses (Währungsumstellung) verändert hat. Wurde die erste extrinsische Bedeutung implizit in einem Modell fixiert, kommt es später zu einer Divergenz in den extrinsischen Bedeutungen zwischen Benutzer und Programm (stellvertretend für den Entwickler vor 20 Jahren) und als folge davon möglicherweise zu falschen Berechnungsergebnissen (oder falschen Interpretationen von Ergebnissen durch den Benutzer)

Modelle über Interpretationen (extrinsische Bedeutungen) haben keine intrinsische Bedeutung.

Jede Repräsentation von Modellierungen einer Interpretation (extrinsischen Bedeutungen) wiederum hat aber keine intrinsische Bedeutung. Ein Benutzer kann daher nicht in jedem Fall vorauserkennen, ob durch ein bestimmtes Modell seine eigenen Bedürfnisse richtig erkannt werden. Dazu müsste er nämlich wieder ein neues Modell über die hinter dem untersuchten Modell stehenden Interpretationen des Entwicklers bilden. Er kann aber lediglich (zunächst) erfolgreich eine systematische Interpretation projizieren. Zum Beispiel weil das System für einige Zeit Sätze erzeugen kann, die als sein Wissen über eine Situation interpretiert werden können und weil es dazu in der Lage ist, Schlüsse, Vorhersagen und Aktionen abzuleiten, die mit seinen eigenen Interpretationen einhergehen. Aufgrund der fehlenden intrinsischen Bedeutung der Modellsymbolik ist diese Projektion aber unvollständig. Das Systemverhalten kann also jederzeit (spontan) deutlich von der eigenen Interpretation des Nutzers abweichen, sobald eine vorher nicht überprüfte neue Situation eintritt. Dabei ist es unerheblich, wie viele andere Situationen vorher positiv überprüft wurden. Dieser Umstand ist in etwa vergleichbar mit dem Versuch, nur über Blackbox Tests Rückschlüsse auf die Korrektheit einer unbekannten Softwarekomponente zu ziehen.

Ein oft genanntes Beispiel in diesem Zusammenhang ist ein System, das alles über einen bestimmten Raum zu wissen scheint, sich dann aber plötzlich so verhält, als ob es der Meinung wäre, alles in diesem Raum würde aufhören zu existieren, sobald man ihn verlässt [78]. Das ist ein Umstand, mit dem das System bisher nicht konfrontiert war, solange die Übereinstimmung mit den eigenen Interpretationen überprüft worden war. Der Bezug zum Frame-Pro-

blem entsteht, da das System aufgrund der in ihm gespeicherten Daten über keine intrinsische Bedeutung verfügt, was sich durch eine Änderung (d.h. neue Informationen) verändert und was nicht und Schlüsse dahingehend nur aus bereits bekannten Situationen ziehen kann.

Die in [78] genannten Kriterien für das Frame-Problem aus der Sicht des Symbol Grounding entsprechen dabei exakt den in 2.4.2 genannten Kriterien für die Registrierung von SUB-Phänomenen. In beiden Fällen tritt das Phänomen überraschend in einem zu einem bestimmten Zeitpunkt korrekten System auf, das in eine unvor(her)gesehene Situation gerät und ein Verhalten erzeugt, das nicht mehr mit der systematischen Interpretation des Beobachters über seine eigenen Bedürfnisse oder die Fähigkeiten der Anwendung übereinstimmt.

Die Ursache für das unvorhersehbare Auftreten von SUB-Phänomenen versteckt sich folglich zum einen in der wesentlichen konzeptionellen Eigenschaft ubiquitärer Systeme (der Kontextadaption), die diese von anderen Softwaresystemen unterscheidet und die auf Modellen veränderlicher Umgebungen (mit veränderlichen extrinsischen Bedeutungen) basiert, zum anderen in der Art ihrer Verwendung (Anpassung an nicht direkt beobachtbare Benutzerbedürfnisse). Beide Faktoren werden in den folgenden Abschnitten noch einmal präzisiert.

3.2.2 Was ist Kontextadaption?

Auswahl von Funktion und Realisierung abhängig von der Situation

In 1.1 war definiert worden, dass sich das wesentliche Merkmal des Ubiquitous Computing (Nutzbarkeit in möglichst vielen Situationen) auf eine wesentliche Eigenschaft ubiquitärer Systeme zurückführen lässt. Diese Eigenschaft ist die Anpassung von Funktionen einer Anwendung und deren Realisierung an ihre Umgebungsbedingungen (Situation), zum Beispiel die verfügbare Hardware oder die momentane Tätigkeit des Benutzers (Kontextadaption).

Vereinfacht gesagt wird durch die automatisierte Auswahl geeigneter Funktion und der Art ihrer Realisierung gegenüber bisherigen Nutzungsformen die Nutzung von Computern und Computeranwendungen auf die maximal mögliche Anzahl von Situationen eines Benutzers ausgeweitet. Für diese automatische Auswahl bedienen sich ubiquitäre Anwendungen der Kontextadaption.

Definition 6: *Kontextadaption*

Kontextadaption ist eine explizite Anpassung des beobachtbaren Verhaltens oder des inneren Zustandes eines Systems an seinen Kontext.

Die Begründung dafür ist einfach. Der Erzielung absoluter Ubiquität gemäß Definition 1, also der Nutzbarkeit einer Anwendung in allen Situationen stehen im Wesentlichen zwei Probleme entgegen:

- Verfügbarkeit: autarke mobile Systeme sind zwar überall und jederzeit verfügbar, zeitlich oder örtlich beschränkte technische Ressourcen bleiben hingegen ungenutzt (kleinster gemeinsamer Nenner). Die Gesamtnutzbarkeit für den Anwender ließe sich aber theoretisch steigern, wenn auch diese mit der Situation des Anwenders beziehungsweise Systems verknüpften Ressourcen in die Anwendungsnutzung mit einbezogen würden.
- Nutzbarkeit: Universell programmierbare Maschinen erfüllt alle Benutzerbedürfnisse, ausgenommen der Bedienbarkeit in Situationen mit eingeschränkter Interaktionsfähigkeit des Benutzers. Kontextfreie Systeme ohne Benutzerinteraktionen sind zwar immer bedienbar, können hingegen überhaupt keine Benutzerbedürfnisse erkennen. Ihr Nutzen ist somit in einem größeren Zusammenhang gesehen mehr oder minder zufällig.

Als Folge der situationsabhängigen Ressourcen für die Verfügbarkeit muss die Anwendung gleichzeitig realisierte Fähigkeiten, das bedeutet Funktionalität und deren Qualität, also die gleichzeitig erfüllbaren Anforderungen möglicherweise situationsabhängig einschränken. Darüber hinaus muss sich die Realisierung der Funktionalität abhängig von den verfügbaren Ressourcen (einschließlich unterschiedlicher Interaktionsfähigkeit des Benutzers) verändern können.

Um die zuvor maximierte Verfügbarkeit auch zu nutzen, muss es daher gelingen, Benutzerbedürfnisse auch mit reduzierten Benutzerinteraktionen in das System mit einzubeziehen, beispielsweise durch Integration von Kontext als eine Form der erweiterten und möglicherweise unbewussten oder indirekten Benutzereingabe.

Folglich muss die Anwendung teilweise ohne Benutzerinteraktion die Benutzerbedürfnisse erkennen und berücksichtigen können, um die situationsabhängige Interaktionsfähigkeit des Benutzers für die Nutzbarkeit zu berücksichtigen. Zudem adaptiert die Anwendung Fähigkeit, Realisierung und

Verhalten an die jeweilige Situation der Anwendung. Dafür reicht es aber aus, die jeweils für den Ablauf der Anwendung gerade relevanten Merkmale der Situation der Anwendung und des Benutzers, also ihren Kontext zu berücksichtigen (=> Kontextadaption).

Definition 7: *Kontext*

Kontext ist die hinreichend genaue Charakterisierung der Situationen eines Systems anhand von für die Adaption dieses Systems relevanten und vom System wahrnehmbaren Informationen.

Kontext können beliebige relevante Informationen sein.

Typische Kontexte sind beispielsweise Beschreibungen von Ort, Zeit und Person einer Anwendungsnutzung [26], soziale Beziehungen zwischen Personen oder Zustände (Auslastung, Speicherplatz) von Hardware [28]. Hier sieht man bereits, dass die Spanne möglicher Umgebungsbedingungen sehr groß sein kann, die meisten aber weder die Art und Weise, in der das Anwendungsprogramm stattfindet, beeinflussen können, noch es bei den meisten Umgebungsbedingungen einen Nutzen bringen würde, sie im Anwendungsprogramm zu berücksichtigen. Die meisten Umgebungsbedingungen sind also nicht relevant bezüglich des "Stattfindens" einer konkreten Anwendung, wie folgendes Beispiel zeigt:

> Für ein tastaturbedientes Textverarbeitungsprogramm ist die Verkehrsdichte auf der Straße unter dem Büro, in dem es eingesetzt wird, in den meisten Fällen völlig irrelevant, ebenso, ob der gegenwärtige Benutzer einen starken Akzent spricht, sein Kühlschrank gerade kaputt gegangen ist, oder wo er am Wochenende seine Freizeit verbringen wird.

Weiterhin sieht man aber auch leicht ein, dass in den meisten Fällen nicht einfach pauschal angeben kann, welche Umweltbedingungen relevant sind und welche nicht.

> Handelt es sich beispielsweise bei oben erwähntem Textverarbeitungsprogramm um eine sprachgesteuerte Anwendung, kann der durch den dichten Verkehr auf der Straße unter dem Bürofenster erzeugte Lärm sehr wohl relevant sein. Im Gegensatz zum vorigen Beispiel kann es hier nützlich sein, diese Umgebungsbedingung im Programm zu berücksichtigen und eine alternative Tastaturbedienung vorzusehen. Auf diese kann dann zurückgegriffen werden, wenn die Erkennungsgenauigkeit durch Lärm oder Akzent des jeweiligen Benutzers unter einen bestimmten Qualitätslevel fällt.

3.2.3 Frame-Problem und Kontextadaption

Definition 7 und andere vergleichbare Definitionen von Kontext (z.B. [26][28][29][30]) legen nahe, dass zwischen den Begriffen *Kontext* und *Situation* ein Zusammenhang, aber auch ein deutlicher Unterschied besteht. Situation wird dabei im Sinne eines universellen Zustandes verwendet, während der Kontext lediglich spezifische Aspekte enthält, die eine Situation charakterisieren [23] oder beschreiben [79].

Es gibt einen Unterschied zwischen Situation und Kontext.

Definition 8: *Situation*

Eine Situation ist die augenblickliche Lage, die Verhältnisse, die Umstände oder der allgemeine (objektive) Zustand, in dem sich jemand (oder etwas) befindet.

Ihren Ursprung hat diese Sichtweise in Forschungen auf dem Gebiet der *Situatedness* [80][81]. Dort wird eine Situation als Ressource für menschliche Kognition und Aktion angesehen. Entscheidend ist, wie die Situation als Ganzes die menschliche Denkweise und Interpretationen beeinflusst, ohne vorher spezifische Aspekte daraus zu isolieren. Die Situation entspricht dort also einer objektiven Wirklichkeit, vor eventuellen Interpretationen durch das menschliche Gehirn. Eine Situation ist folglich eine beobachterunabhängige potenziell unbegrenzte Ressource, die inhärent offen gegenüber (Re-)Interpretationen ist [23].

Aufgrund des Symbol Grounding Problems (siehe 3.1) lassen sich Situationen aber nicht durch ein Symbolsystem mit intrinsischer Bedeutung beschreiben. Folglich gibt es keine allgemeine vollständige objektive Beschreibung einer Situation. Statt dessen müssen Situationen als Ausdruck einer bestimmten Interpretation (*ungrounded* mit Bezug auf eine extrinsische Interpretation) beschrieben werden und sind damit beobachterabhängig und im Gegensatz zu der beschriebenen Situation nicht mehr offen für Reinterpretation. Im Fachgebiet der Entwicklung von Softwaresystemen, welche die Situation eines Benutzers oder allgemeiner der Umgebung einer Anwendung berücksichtigen, hat sich für eine bessere Unterscheidung zwischen Situation und ihrem Modell folglich der Begriff *Kontext* etabliert. Im allgemeinen Sprachgebrauch werden die beiden Begriffe jedoch oft synonym verwendet, obwohl die eigentliche Sprachbedeutung von Situation und Kontext [3][4] durchaus eine äquivalente Unterscheidung (Zustand vs. Zusammenhang) zulässt.

Diese Art der Festlegung des Kontextbegriffes impliziert leider, dass der Entwickler einer kontextadaptiven Anwendung festlegen muss, dass bestimmte Aspekte wie "Orte, Identitäten und Zustände von Personen, Gruppen und

Kontext ist ein Modell der Situation

Berechnungs- und physikalischen Objekten" [26] signifikant sind, während andere Aspekte keine Relevanz besitzen. Die Bedeutung der in eine konkrete Kontextbeschreibung aufgenommenen Aspekte ist daher mehr oder weniger festgelegt [23] durch die extrinsischen Interpretationsbezüge auf die persönliche Wahrnehmung und Erfahrungen des Entwicklers. In manchen Fällen kann sie sogar überinterpretiert [78] sein, wenn aufgrund bestimmter Annahmen (z.B. alle geschlossenen Türen blockieren die Sicht) eine Menge von Situationen gemeinsam abstrahiert werden, die bei Nichtzutreffen der Annahmen (z.B. Türen mit Glaseinsätzen erlauben das Eindringen von Licht in dahinter liegende Räume) unterscheidbar sind. Genauso gut könnten also noch weitere möglicherweise relevante Aspekte einer Situation in ihre Beschreibung (den Kontext) aufgenommen oder nicht aufgenommen werden, ebenso wie weitere möglicherweise irrelevante Aspekte. Diese Offenheit einer Situation gegenüber Reinterpretation hat eine wichtige Bedeutung, da individuelle Benutzer sich dafür entscheiden könnten, bestimmten Aspekten der Umgebung eine (andere) Signifikanz zuzuweisen, die zuvor (im Modell) nicht vorhanden war [23].

Dieses Problem der Festlegung relevanter Aspekte eines Situations- oder Weltmodells entspricht genau dem in 3.1 beschriebenen Ursachen des Frame-Problems und (mit 3.2.1) auch der Ursache des SUB-Phänomens. SUB-Phänomene entstehen in ubiquitären Anwendungen also durch die Verwendung von Situationsmodellen für die Adaption (Kontextadaption).

Ursache und Wirkung werden dagegen vom Benutzer über die augenblickliche Situation wahrgenommen. Da die Situation aber offen gegenüber Reinterpretationen ist, kann die Wahrnehmung des Benutzers sich von der durch das Situationsmodell Kontext ausgedrückten Wahrnehmung des Entwicklers unterscheiden. Dies ist umso mehr der Fall, wenn ein zeitlicher Abstand zwischen beiden gegeben ist, in dem sich die Zusammenhänge der Wirklichkeit verändern können.

3.2.4 Verstärkungseffekte

Der alleinige Umstand, dass ubiquitäre Systeme Kontextadaption und damit explizite Modelle ihrer Nutzer oder allgemeiner der Umgebungssituation verwenden, erklärt zwar das Auftreten von Frame-Problemen in Gestalt von SUB, nicht jedoch den von prototypischen Versuchen vermittelten Eindruck ihrer (überproportionalen[1]) Häufigkeit.

[1] bezogen auf die Anzahl der im Programm unterschiedenen Situationen

Häufung von SUB im UC schwer zu belegen

Bereits aber bei dem Versuch einer vergleichbaren Quantifizierung von SUB-Phänomenen, um damit die Vermutung der überproportionalen Häufigkeit zu belegen, entsteht ein Problem. Schließlich lassen sich prinzipiell keine nichtubiquitären Anwendungen erstellen, welche die gleiche Funktionalität und Aufgabe besitzen, wie ein ubiquitäres System. Ansonsten bestünde die einfachste Lösung des SUB-Problems darin, ubiquitäre Anwendungen auf nichtubiquitäre Weise zu erstellen. Häufigkeitsvergleiche können daher nur qualitativ erfolgen, beispielsweise durch Vergleich der Anzahl erfolgreicher nichtubiquitärer Anwendungen mit der Anzahl von ubiquitären Projekten, die wegen SUB die Erwartungen nicht erfüllen konnten. Dies ist aber fast nur im Rahmen eigener Erfahrungen möglich (womit man wieder bei der anfänglichen Vermutung wäre), da es zum einen nicht besonders viele real existierende ubiquitäre Anwendungen gibt, zum anderen gescheiterte Projekte selten (wahrheitsgemäß) veröffentlich werden. Neben der persönlichen Erfahrung lässt sich aber noch ein weiteres Indiz aufspüren, welches die Vermutung stützen kann, dass nicht alleine die höhere Komplexität ubiquitärer Anwendung an der erhöhten Wahrnehmung von SUB schuld sein kann. Das gehäufte Auftreten eines einzigen Szenariotyps in vielen Publikationen (nämlich die adaptive Informationsanzeige) deutet indirekt auf Schwierigkeiten mit anderen ubiquitären Anwendungsszenarien hin. Die einzige prinzipielle Schwierigkeit, die in eigenen Versuchen mit Prototypen auftrat, die keine reinen kontextadaptiven Informationsanzeigen waren, war aber das SUB-Phänomen, während kontextadaptive Informationsanzeigen weitgehend von einer Wahrnehmung des Problems verschont blieben.

Häufung von SUB im UC schwer zu belegen

Daraus kann man folgern, dass außer der Tatsache der Verwendung von Weltmodellen und ihrer Komplexität noch weitere Faktoren existieren, die dazu führen, dass SUB in manchen ubiquitären Anwendungen stärker und in anderen weniger auftritt, beziehungsweise empfunden wird. Ausgehend von dieser Annahme konnten zwei solcher Verstärkungsfaktoren identifiziert werden.

Perzeption

Der erste Faktor bezieht sich lediglich auf die Perzeption. Frame-Probleme werden nur dann zu SUB, wenn ihre Auswirkungen auch tatsächlich beobachtet werden können. Denn die Tatsache, dass ein System falsche Schlussfolgerungen aus einer Situation zieht, führt nur dann zu einem SUB, wenn die Auswirkungen von einem Benutzer bewusst wahrgenommen werden können. Dies erklärt auch, warum die meisten kontextadaptiven Systeme sich lediglich mit der Anzeige von Informationen beschäftigen.

Ubiquitäre Anwendungen unterscheiden sich in der Wahrnehmungshäufigkeit von SUB.

Eigene Versuche am Prototyp I haben nämlich gezeigt, dass selbst die völlig zufällige Einblendung von verschiedenen Informationsseiten (Wetter, Zugverspätung, Nachrichten, etc.) auf einem Umgebungsterminal von den Nutzern als positiv (bzw. deren Abwesenheit als negativ) bewertet wurde, zumindest so lange die Informationen an sich nicht falsch waren (falscher Wetterbericht etc.). Informationen, die nicht den Nutzerbedürfnissen entsprachen, wurden einfach ignoriert und daher nicht als SUB registriert. Daher ist es nicht verwunderlich, dass viele publizierten Prototypen ([21][22][27][56][57] usw.) ubiquitärer Anwendungen ein ähnliches, bezogen auf die leichte Ignorierbarkeit von SUB günstiges Szenario implementieren.

Sind die Auswirkungen falscher Schlüsse dagegen direkt über die physikalische Umgebung erfahrbar (Dunkelheit, Temperatur, etc.) können sie wesentlich schlechter ignoriert werden. Ubiquitäre Anwendungen besitzen also ein unterschiedliches Potenzial an Störfaktoren, welche die Wahrscheinlichkeit beschreiben, dass ein bestimmtes Fehlverhalten auch tatsächlich als SUB registriert werden kann.

Fehlende Kompensation

Die zweite Gruppe von Verstärkungsfaktoren des SUB-Phänomens in ubiquitären Systemen sind die gegenüber nichtubiquitären Systemen reduzierten oder fehlenden Kompensationsmechanismen.

Dass eine Kompensation von Frame-Problemen und damit auch SUB möglich ist, macht man sich leicht mithilfe der folgenden Überlegung klar. Auch Menschen können dem Frame-Problem unterliegen. Entweder wenn sie eine Interpretation über die Interpretation eines anderen herstellen (darüber denken, was ein anderer denkt, brauchen könnte oder wie er reagieren wird) oder für die Bewertung einer Situation nicht auf direkte persönliche sensorische Informationen, sondern auf eine Beschreibung in Form einer (ungrounded) Formalisierung zurückgreifen müssen (Missverständnisse bei der Interpretation von Zeugenaussagen, Büchern, Sprache etc.). Zur Vermeidung des Frame-Problems und damit SUB ("ich habe gedacht, du hättest gedacht" Phänomene) sind daher verschiedene Verfahren entwickelt worden, unterschiedliche Interpretationen einer Situation miteinander abzugleichen (Rückfragen, mathematische Beweise etc.).

Nutzer können SUB durch Mediation zwischen System und Umgebung an der direkten Benutzerschnittstelle kompensieren.

Auch nichtubiquitäre Systeme besitzen einen Kompensationsmechanismus, der SUB-Phänomene von vornherein verhindert, oder zumindest ihre Wiederholung vermeidet. Dieser kann auch als eine Form der Beeinflussung des Systemverhaltens von außen betrachtet werden, die zwischen Umgebung und System angesiedelt ist. Eine solche manuelle Mediatorfunktion zwischen ei-

nem System und seiner Umgebung wird in [20] im Rahmen des *Requirements-Engineering* als *Spontane Hülle* bezeichnet. Diese wird zwischen einer Umgebung und dem geplanten Kern eines Systems positioniert.

Definition 9: *Spontane Hülle*

Die Spontane Hülle definiert Aufgaben eines menschlichen Benutzers, die eine begrenzte Anpassung des Systems an die jeweilige Einsatzsituation enthalten.

Zu den typischen Aufgaben einer solchen spontanen Hülle gehören beispielsweise:

- Filtern relevanter Ereignisse,
- Ereignisse aktiv herbeiführen, auf die das Programm reagieren kann,
- Übersetzung von Umgebungsanforderungen/Informationen,
- Ausführung von Aktivitäten, die das Programm nicht oder nur unzureichend leisten kann,
- Spezialfälle behandeln, deren Planung nicht wirtschaftlich ist (weil beispielsweise zu selten),
- Informelle Kommunikation mit der Umgebung (zum Beispiel Beratung).

Dies stellt in einem gewissen Rahmen nichts anderes dar, als eine manuelle Anpassung des Systems an die Umgebung durch den Benutzer. Benötigt ein Programm beispielsweise die Umgebungstemperatur und kann die Existenz eines bestimmten Sensors dafür nicht zum Zeitpunkt der Implementierung festgelegt werden, dann kann diese Aufgabe, das heißt das Suchen eines Thermometers, das Ablesen eines Wertes und dessen Weiterleitung an das Programm auf den Benutzer in Form einer Eingabe über die Benutzerschnittstelle abgewälzt werden. Der Benutzer fungiert der Anwendung gegenüber verborgen als Vermittler der Information und kann flexibel auf Änderungen der Situation (Verfügbarkeit von Thermometern, Umstellung von Fahrenheit auf Grad Celsius usw.) reagieren. Genauso braucht ein nichtubiquitäres System keine Berechnungen über mögliche Benutzerbedürfnisse anstellen. Es ist Aufgabe des Benutzers, seine Bedürfnisse über die Benutzerschnittstelle in geeigneter Weise zu formulieren, so dass sie der Interpretation des Entwicklers entsprechen und korrekt umgesetzt werden können.

Im Ubiquitous Computing entfällt diese Kompensationsmöglichkeit ganz oder teilweise, weil die Spontane Hülle und die darin geforderten Interaktionsfähigkeiten des Benutzers normalerweise den begrenzenden Faktor für

Situationen darstellt, in denen ein System genutzt werden kann. Je kleiner die Spontane Hülle, desto vielseitiger ist eine Anwendung einsetzbar, desto weniger Möglichkeiten bestehen aber auch, die möglicherweise unterschiedlichen Interpretationen einer Situation zwischen Nutzer und (stellvertretend für den Entwickler) Programm auszugleichen. Dadurch kann die Zahl der SUB-Phänomene aber stark ansteigen, besonders wenn sich die Wirklichkeit zwischen dem Zeitpunkt der Entwicklung eines Systems und seiner Verwendung etwa wegen einer langen Einsatzdauer stark verändert hat.

3.2.5 Zusammenfassung

Kontextadaption ist ein Konzept, die ubiquitäre Nutzbarkeit einer Anwendung in möglichst vielen Situationen zu gewährleisten. Zu diesem Zweck ersetzt (automatisiert) die Kontextadaption einen Teil der Spontanen Hülle des Systems durch ein Welt- oder Benutzermodell, das prinzipiell dem Frame-Problem unterliegt.

Gleichzeitig bedeutet eine Verkleinerung der Spontanen Hülle eine Verringerung der direkten zeitnahen Interaktionen zwischen Benutzer und System. Dies verhindert eine manuelle Angleichung möglicherweise unterschiedlicher Interpretationen der Wirklichkeit zwischen Nutzer und Programm und verhindert damit eine Kompensation der durch das Frame-Problem verursachten SUB-Phänomene. Vereinfacht gesagt ist es dem Benutzer bei einer teilweise selbstständig im Hintergrund unsichtbar agierenden Anwendung nicht mehr ohne weiteres möglich, sich auf deren Unzulänglichkeiten einzustellen.

Das ist auch der Grund, warum das Frame-Problem zuallererst bei der Erforschung weitgehend autonom agierender Systeme (Robotern) offensichtlich wurde.

3.3 Kalibrierung als Lösungsansatz

Die Betrachtung der Ursachen von SUB hat deutlich gemacht, dass Modelle der Wirklichkeit ganz generell veralten können, indem bestimmte Annahmen, auf denen die Modellabstraktion basiert, ihre Gültigkeit verlieren können. Aufgrund des als prinzipiell anzusehenden Frame-Problems ist es zudem unmöglich, solche Divergenzen aus dem Modell heraus selbst oder auch von außen vollständig zu erkennen. Zum einen, weil Modelle der Wirklichkeit in der Regel unendlich viele implizite Annahmen enthalten. Zum anderen aber auch, weil Modellrepräsentationen keine extrinsischen Bedeutungen enthalten können und daher Modelle über extrinsische Modellinterpretationen

anderer Personen aufgrund des Perspektivunterschiedes nicht ohne weiteres auf ihre Richtigkeit überprüft werden können.

Folglich lässt sich kein Modell der Wirklichkeit erstellen, das für alle Zeiten die Wirklichkeit korrekt abbildet, oder sich selbst zu korrigieren vermag, insbesondere wenn dabei Aussagen über andere Modellinterpretationen wie Nutzerbedürfnisse gemacht werden sollen. SUB-Phänomene können also nicht gänzlich verhindert werden. Ihre Zahl und Auswirkungen können aber durch die richtige Vermeidungsstrategie reduziert werden. Dazu muss ein zu nichtubiquitären Anwendungen vergleichbarer Kompensationsmechanismus entwickelt werden, der allerdings nicht ausschließlich auf zeitnahen Interaktionen zwischen Benutzer und Programm basieren darf. Ansonsten könnten die in Definition 1 genannten Ubiquitätsbedingungen verletzt werden. Die Grundvoraussetzungen eines solchen Lösungsansatzes sind dabei:

- Der Benutzer benötigt den Zugriff auf das Interpretationsmodell des Entwicklers, das in Form der Systemspezifikation vorliegt.
- Umgekehrt muss der Benutzer ein Modell seiner eigenen Interpretationen erstellen können, das vom System verarbeitet werden kann.
- Beide Interpretationsmodelle müssen in jeweils verständlicher Form vorliegen, das heißt, es muss eine gemeinsame Interpretationssystematik in Form von extrinsischen Bedeutungen vorliegen.
- Beide Modelle müssen ständig veränderbar sein, beispielsweise indem das System das Nutzermodell in sich aufnehmen und der Benutzer sein eigenes Modell jederzeit aktualisieren kann.

Unter diesen Voraussetzungen kommen verschiedene prinzipielle Lösungsansätze in Frage.

3.3.1 Direkte Programmierung des Systems als erster Lösungsansatz

Die manchmal in Form von Script-Programmierung (z.B. [53]) oder vollständiger Reprogrammierbarkeit durch Auslieferung des Sourcecodes (z.B. [54]) praktizierte direkte Programmierung eines Systems ist dabei ein solcher Kompensationsmechanismus, der nicht notwendigerweise mit den Zielen der Ubiquität in Konflikt steht. Da jedoch eine Live-Programmierung durch das Ziel der ubiquitären Nutzung und der dahingehend verkleinerten Spontanen Hülle (möglichst wenig Interaktionen) ausgeschlossen ist, können nach einer bestimmten Zeit aufgrund des Frame-Problems SUB Ereignisse entstehen, obwohl die Interpretationen am Anfang (Entwickler=Nutzer) identisch sind (siehe auch Beispiel in 3.1).

Allerdings kennt der Benutzer die Interpretation des von ihm erstellten Programms und kann daher viele kritische Situationen mit einem gewissen zeitlichen Vorsprung vorhersehen und durch eine vorausschauende Reprogrammierung des Systems verhindern.

In 2.4.3 wurde als Beispiel ein SUB des dort zuvor beschriebenen Prototyps genannt, das sich auf eine falsche Erkennung der Urlaubsabwesenheit bezog. Ausgelöst wurde das SUB durch ein überlanges Ausbleiben der Bewohner an einem Silvesterabend.

Eine genaue Kenntnis der Bedingungen für ein Erkennen der Situation "Urlaub" (wenn sich von 22:00 bis 6:00 des darauf folgenden Tages niemand in der Wohnung aufhält) würde es den Benutzern erlauben, diese kritische neue oder Ausnahmesituation vorherzusehen und das System entsprechend zu reprogrammieren (nicht an Silvester, Positivliste von Tagen, an denen keine Situation "Urlaub" eintreten kann, vorübergehende Deaktivierung der Funktion etc.).

Kann ein potenzielles SUB dennoch einmal nicht vorhergesehen werden, so kann zumindest seine Wiederholung verhindert werden. Dabei ist die möglichst vollständige Reprogrammierung von großer Bedeutung. Ein beispielsweise bloßes Hinzufügen einer neuen Ausnahme oder Teilregel kann zwar ein SUB effektiv verhindern, ändert aber möglicherweise nichts an der generell falschen Interpretation einer Situation [78] oder erzeugt sogar noch eine größere Anzahl von Folgeproblemen (z.B. führt der Ausschluss der Erkennung an Silvestertagen möglicherweise zu unerwünschten Nichterkennungen von Urlaub, wenn einmal Silvester tatsächlich ein Urlaub gemacht wird). Beispiele für solche Folge-SUBs waren in 2.4.3 in Zusammenhang mit der Einführung einer Semiautomatik für die Lichtsteuerung genauer beschrieben worden.

Direkte Programmierung ist ein erfolgsversprechender aber zu komplizierter Ansatz.

Einige Probleme bleiben bei der Verwendung der direkten Programmierung als Kompensationsmöglichkeit von SUB-Phänomenen allerdings ungelöst:

- Nutzen mehrere Benutzer die gleiche Instanz eines Systems (wie im Beispiel des Prototyps aus 2.3), ergeben sich Schwierigkeiten bei der gleichzeitigen Reprogrammierung für die Bedürfnisse mehrerer Personen.
- Die Reprogrammierung ist zeitaufwändig und erfordert besondere Fähigkeiten. Dies schränkt den Nutzerkreis erheblich ein.
- Die Komplexität der Reprogrammierung steigt mit zunehmender Anzahl ubiquitär nutzbarer Funktionalität stark an. Dadurch wird es selbst für den geübten Entwickler schwierig, eine bestimmte Interpretation des Systems und damit kritische Situation vorherzusagen. Selbst in ein-

facheren Systemen wie dem in 2.3 beschriebenen Prototyp I war es auch bei bereits mehrfach aufgetretenem SUBs in manchen Fällen sehr schwierig, die genaue Ursache der unterschiedlichen Interpretation (vgl. einem Debugging) einzugrenzen. Die Vorhersage von SUB wird dadurch erst recht unmöglich.

3.3.2 Andere bekannte Kalibrierungsansätze und ihre Probleme

Neben der direkten Programmierung gibt es noch benutzerfreundlichere Ansätze, das Frame-Problem und die damit einhergehenden SUB-Phänomene in kontextadaptiven Anwendungen zu kompensieren. Diese sind jedoch im Gegensatz zur direkten Programmierung mit prinzipiellen Einschränkungen behaftet.

Benutzerpräferenzen und Konfiguration

Das bekannteste Mittel, einer Anwendung zeitlich versetzt Benutzerbedürfnisse mitzuteilen, ist die Konfiguration von Benutzerpräferenzen. Dabei werden durch den Benutzer in der Regel nach dem ersten Start einer Anwendung bestimmte Festlegungen getroffen, die bis zu einer erneuten Änderung ihre Gültigkeit behalten. Über Benutzerpräferenzen ließen sich so theoretisch auch unterschiedliche Interpretationen einer Situation angleichen.

Das Problem der Benutzerpräferenzen ist aber ihre allgemeine Natur, das heißt, sie bieten lediglich eine generelle Auswahl oder Parametrisierung von alternativen Verhaltensweisen, die zum Entwicklungszeitpunkt durch den Programmierer festegelegt wurden. Bezüglich des Frame-Problems bringt das aber kaum einen Vorteil. Genauso gut ließen sich eine größere Anzahl von Variationen einer Anwendung erstellen, unter denen der Benutzer zunächst eine auswählen kann, die seiner Interpretation bestimmter Situationen am nächsten kommt. Das Frame-Problem tritt trotzdem auf, da bei dessen Definition bereits davon ausgegangen worden war, dass die Interpretation im System und die des Benutzers für einen gewissen Zeitraum übereinstimmten (siehe 3.1) und erst später spontan divergieren.

Konfigurationen sind nicht zwischen verschiedenen Funktionen und Realisierungen übertragbar.

Zudem integrieren ubiquitäre Systeme oft eine Reihe von Einzelanwendungen, zwischen denen die Kontextadaption dann situationsabhängig auswählt. Benutzerpräferenzen sind aber nicht ohne weiteres zwischen den einzelnen Anwendungen übertragbar und müssten daher bei jeder größeren Adaption (neue Funktionen/Realisierungen) des Gesamtsystems erneut in der durch die jeweilige Implementierung verlangten Art eingegeben werden.

Lernfähige Systeme

Lernfähige Systeme verschieben SUB und FP in den Lernalgorithmus

Lernfähige Systeme basieren auf dem Ansatz, SUB-Phänomene im Nachhinein zu erkennen, beispielsweise durch eine Beobachtung des Nutzers und seiner Zufriedenheit. Daraus wird dann eine Strategie für die Vermeidung deren zukünftigen Auftretens abgeleitet. Dabei gibt es aber gleich mehrere wesentliche Probleme:

- Solche Mechanismen können lediglich bereits aufgetretene SUBs verhindern.
- Das Frame-Problem wird genau genommen nur aus der Anwendung heraus in den Lernalgorithmus verlagert. Für die Erkennung von SUBs und von Maßnahmen zur Verhinderung eines erneuten Auftretens sind aber möglicherweise wesentlich komplexere Weltmodelle notwendig, als für die Anwendung selbst. Damit steigt das Risiko für SUB sogar an.
- Das bloße Hinzufügen einer neuen Ausnahme oder Teilregel kann zwar ein SUB effektiv verhindern, ändert aber möglicherweise nichts an der generell falschen Interpretation einer Situation [78] oder erzeugt sogar noch eine größere Anzahl von Folgeproblemen.

3.3.3 Modellbasierte Kalibrierung der Kontextadaption

Vereinfachung des Reprogrammierungsansatzes.

Lernfähige Systeme eignen sich wegen des letztgenannten Nachteils also höchstens als eine Ergänzung zur direkten Programmierung, genauso wie Benutzerpräferenzen. Der in dieser Arbeit entwickelte Kalibrierungsmechanismus für Kontextadaption greift daher den einzig wirksamen Ansatz der direkten Programmierung als Kompensation des Frame-Problems auf.

Dessen Nachteile sind im Gegensatz zu den anderen genannten Verfahren nämlich nicht prinzipieller Natur und können durch geeignete Techniken beseitigt werden.

Der daraus entstehende neue Kalibrierungsansatz behält die doppelte Vermeidungsstrategie der direkten Programmierung bei, die aus einer Vorauserkennung von SUB kritischen Situationen durch den Benutzer sowie einer (zeitversetzt) flexiblen manuellen Beseitigungsmöglichkeit durch völlige Reprogrammierbarkeit des Systems besteht.

Für eine Verbesserung der Nutzerfreundlichkeit greift die Kalibrierung auf einige Ideen der Benutzerpräferenzen und der Lernfähigkeit zurück und überträgt sie auf die direkte Programmierung. Dadurch werden deren Vorteile übernommen, die prinzipielle Tauglichkeit des Kompensationsmechanismus bleibt aber erhalten.

Wie bei der Lernfähigkeit, wird das Frame-Problem (FP) zunächst aus der Anwendung oder den einzelnen Anwendungsfunktionen heraus in ein vom Rest der Anwendungen unabhängiges explizites generisches Modell (Teilsystem) der Kontextadaption verlagert. Dies erreicht man, indem man alle FP-kritischen Bestandteile, also die Nutzer- oder allgemein Situations- und Weltmodelle aus den einzelnen Anwendungsfunktionen herauslöst und zu einem gemeinsamen generischen Modell vereinigt. In den Anwendungsfunktionen (dem Systemkern) verbleiben dann lediglich die Realisierungen von Aktionen der Anwendung. Die FP-kritischen Entscheidungen, welche Funktion wann mit welchen Parametern zu aktivieren ist, wird in das Kontextadaptionsmodell ausgelagert.

Das Kontextadaptionsmodell selbst wird vollständig reprogrammierbar ausgelegt. Das bedeutet, es besteht nicht nur aus einem fest definierten Modell, in dem lediglich bestimmte Regeln hinzugefügt oder verändert werden können. Statt dessen kann auch die Art und Realisierung des Modells zur Laufzeit verändert, also reprogrammiert werden. Ein solches Modell wird im Folgenden kalibrierbares Modell (kurz K-Modell) genannt.

Da sich eine generische Kontextadaption mit wenigen Grundelementen (Sensoren, Interpreter und Aktuatoren) beschreiben lässt, reduziert sich die Schwierigkeit der Programmierung, die notwendigen Hilfsmittel vorausgesetzt, gegenüber einer beliebigen Programmierbarkeit in Script- oder Codeform erheblich. Geeignete Hilfsmittel sind beispielsweise vergleichbar den Nutzerpräferenzen vorgefertigte Bausteine für Sensoren, Interpreter und Aktuatoren, die vom Benutzer nur noch kombiniert werden müssen. Um eine vollständige Reprogrammierbarkeit zu erreichen, ist es natürlich wichtig, dass diese Bausteine auch zur Laufzeit aus externen Quellen importiert werden können.

Eine einfache und intuitiv verständliche Beschreibungstechnik für dieses Vorgehen vorausgesetzt, wirkt diese dabei gemäß den beiden Vermeidungsstrategien (Vorauserkennung/Verhinderung) in beide Richtungen. Das System kann seinen Benutzer mithilfe einer einfachen Beschreibung über derzeit im System gültige Interpretationen von Situationen informieren. Dieser kann dann umgekehrt über eine geeignet veränderte Beschreibung die Situationsinterpretationen mit seinen eigenen Vorstellungen reprogrammieren.

Gegenüber den manuellen Kalibrierungsmöglichkeiten ist deren Automatisierung (z.B. Lernfähigkeit) lediglich als eine komfortable Ergänzung zu verstehen. Wie bereits zuvor diskutiert, trägt die Lernfähigkeit günstigstenfalls nichts zur Kompensierung des Frame-Problems bei, sondern verlagert lediglich das Problem. Diese Verlagerung kann aber auch gezielt verwendet werden, um eine höhere Benutzerfreundlichkeit zu erzielen, solange gewährleis-

tet ist, dass am Ende der Kette eine manuelle Kalibrierung (z.B. durch Austausch des verwendeten Lernalgorithmus) erhalten bleibt. Dadurch wird aber gleichzeitig die Vorhersagbarkeit FP-kritischer Situationen verschlechtert, so dass die Anwendbarkeit einer automatisierten Kalibrierung auf Einzelfälle beschränkt bleiben sollte. Ein einfaches Beispiel dafür wäre, auf diese Weise eine Mehrbenutzerfähigkeit der Kalibrierung herzustellen. Das System könnte lernen, die einzelnen manuellen Kalibrierungen unterschiedlichen Benutzersituationen (A, B, AB) zuzuordnen und so bei einem Benutzerwechsel die von einem Benutzer getätigten Kalibrierungen zu entfernen und durch die gespeicherten Kalibrierungen des anderen Benutzers zu ersetzen.

3.3.4 Zusammenfassung

Ubiquitäre Anwendungen unterscheiden sich von normalen Computeranwendungen dadurch, dass sie in einem weit größeren Bereich von Situationen eines Benutzers verfügbar und nutzbar sein müssen. Insbesondere betrifft dies Situationen mit unterschiedlicher Hardware, unterschiedlichen Bedürfnissen sowie unterschiedlichen und vor allem eingeschränkten Interaktionsfähigkeiten des Benutzers.

Im Unterschied zu normalen Computeranwendungen bedingen ubiquitäre Anwendungen daher nicht zwingend eine direkte und explizite Aufmerksamkeit erfordernde Interaktion. An deren Statt kann im Ubiquitous Computing teilweise eine zweite Interaktionsform, die Kontextadaption treten. Bei der Kontextadaption interagiert der Benutzer indirekt und größtenteils unbewusst über die physikalische (Gegenstände, Bewegungen etc.) und virtuelle Umgebung (z.B. Daten über den Benutzer) mit der Anwendung. Auf diese Weise können ubiquitäre Anwendungen auch im Hintergrund während anderer Tätigkeiten (Auto fahren, Besprechungen usw.) genutzt werden, ohne dass eine explizite Konzentration und Kontrolle des Benutzers erforderlich wäre. Dies kann so weit gehen, dass die Anwendung auch ohne explizite Anwesenheit des Benutzers auf Ereignisse der Realität in seinem Sinne reagiert.

Aufgrund einer sich verändernden Wirklichkeit können in Softwaresysteme verwendete Modelle der Realität aber von der Wirklichkeit abweichen, da bestimmte Modellierungsannahmen (z.B. für Anforderungen) ihre Gültigkeit verlieren können. Dies gilt insbesondere für die Kriterien, aus denen wechselnde Benutzerbedürfnisse erkannt werden.

In normalen Anwendungen kann eine solche Divergenz zwischen Benutzerbedürfnissen und Anforderungen durch die direkte Interaktion (Spontane Hülle, primäre Interaktion) kompensiert werden. Der Benutzer fungiert als Mittler zwischen System und Umgebung und kann so Änderungen der Wirk-

lichkeit einschließlich seiner eigenen Bedürfnisse in für die betreffende Anwendung verständliche Eingaben übersetzen, respektive Ausgaben dahingehend nachbearbeiten. Dies schließt auch mit ein, dass der Benutzer seine eigenen Bedürfnisse den von der Anwendung erfüllbaren Anforderungen unterordnet, oder in Ausnahmesituationen auf eine Benutzung verzichtet.

Der Benutzer aktiviert eine Anwendung mit Sprachausgabe. Aufgrund plötzlich einsetzenden Umgebungslärms erhöht er seine eigene Sprechlautstärke.

Durch die indirekte Interaktion im Hintergrund (Kontextadaption, sekundäre Interaktion) ist diese Art der Kompensation in ubiquitären Anwendungen nicht mehr in allen Fällen möglich. Durch die teilweise unbewusste Nutzung und automatische Adaption des Systems bleiben die erfüllbaren Anforderungen dem Benutzer zudem verborgen und es besteht keine Möglichkeit, die Benutzerbedürfnisse den tatsächlichen Möglichkeiten unterzuordnen. Die nicht mehr kompensierten Divergenzen zwischen Annahmen des Systems an die Umgebung und der Realität, insbesondere zwischen Anforderungen und Benutzerbedürfnissen äußern sich so aus Sicht des Benutzers in spontanem unerwartetem Fehlverhalten (SUB).

Die Anwendung mit Sprachausgabe wird aktiviert, sobald ein Benutzer den Raum betritt. Aufgrund plötzlich einsetzenden Umgebungslärms wäre eine Sprachausgabe aber nicht zu hören. Der Benutzer ist gerade mit anderen Tätigkeiten beschäftigt und ist sich dieses Problems nicht bewusst. Selbst wenn er sich bewusst wäre, weiß er nicht, ob die Anwendung aufgrund des Lärms die Lautstärke erhöht hat, oder nicht. War das in den Anforderungen nicht vorgesehen, kann der Benutzer eine auftretende Meldung nicht verstehen und registriert das Ganze als Fehlverhalten (SUB).

Für die Kompensation von Divergenzen zwischen Modellannahmen und Wirklichkeit und dem daraus entstehenden spontanen Fehlverhalten in ubiquitären Anwendungen wird daher in dieser Arbeit eine dritte Interaktionsform (Kalibrierung, tertiäre Interaktion) entwickelt. Diese basiert wiederum auf einer direkten und bewussten Interaktion, allerdings nicht mit den Anwendungsfunktionen, sondern lediglich mit der Kontextadaption, also den unbewussten im Hintergrund ablaufenden Interaktionsmöglichkeiten. Auf diese Weise können die Modelldivergenzen zeitlich entkoppelt kompensiert werden, ohne die Ubiquität, also die Nutzbarkeit in Situationen zu stören, in denen keine direkten bewussten Interaktionen möglich sind. Allerdings findet die Kompensation nicht mehr wie im Fall der Spontanen Hülle kontinuierlich

statt, sondern unter Umständen nur in diskreten Abständen, um die Wiederholung eines bereits aufgetretenen oder das Eintreten eines sich abzeichnenden SUBs zu verhindern.

Nach dem Auftreten des SUB-Phänomens ändert der Benutzer mithilfe der Kalibrierung die Adaptionsregeln so, dass sie in Zukunft die Umgebungslautstärke selbstständig berücksichtigen.

Die folgende Abbildung zeigt noch einmal die drei Interaktionsmöglichkeiten eines ubiquitären Systems mit kalibrierbarer Kontextadaption:

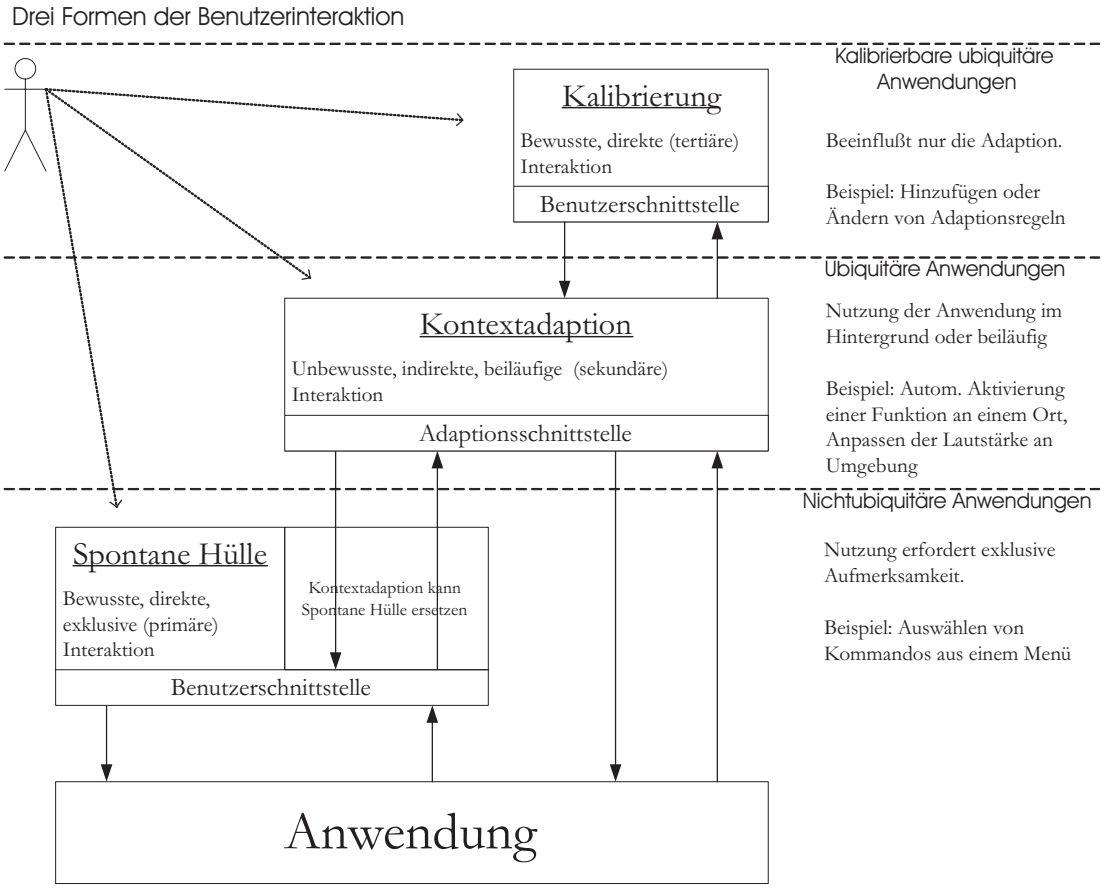


Abb. 13: Drei Formen der Benutzerinteraktion in ubiquitären Anwendungen

Wichtig ist an dieser Stelle auch noch einmal die Besonderheit der Unterscheidung zwischen der Kalibrierung und einer vollständig durch den Benutzer reprogrammierbaren Anwendung (z.B. Open Source Software).

Anstatt die komplette Anwendung zu reprogrammieren, wird bei ersterer lediglich das Teilsystem der Kontextadaption modifiziert. Aufgrund der später in Kapitel 5 gezeigten einfachen generischen Abstrahierbarkeit von Adaption

ist die Kalibrierung wesentlich einfacher zu formulieren als die direkte Änderbarkeit eines beliebigen Programms. Der Schwierigkeitsgrad der Bedienung von Kalibrierung kann daher bei richtiger Ausgestaltung trotzt Generizität, also der Unabhängigkeit der Schnittstelle von einer spezifischen Anwendung, etwa auf dem Niveau von bisher üblichen Anwendungsschnittstellen liegen. Im Vergleich zu diesen kann die Benutzung der Kalibrierung aber zeitlich versetzt zu den in ubiquitären Anwendungen häufig vorkommenden unbewussten Interaktionen und damit ohne Verletzung der in Definition 1 genannten Ubiquitätsbedingungen erfolgen.

KAPITEL 4 Kernprobleme und Anforderungen der Kalibrierung

Was sind die wichtigsten Probleme im Entwurf einer kalibrierbaren Kontextadaption?
Welche Anforderungen ergeben sich daraus?

Dieses Kapitel beschreibt die wichtigsten Probleme und die sich daraus ergebenden Anforderungen an den Entwurf und die Anwendungsmethodik eines Kalibrierungsmechanismus für Kontextadaption, der dazu geeignet ist, das Frame-Problem in ubiquitären Systemen zu kompensieren:

- Wie kann eine vollständige Reprogrammierbarkeit der Kontextadaption gewährleistet werden, ohne eine der beiden Ubiquitätsdimensionen (Verfügbarkeit, Nutzbarkeit) zu behindern?
- Was sind die wichtigsten Probleme, die sich aus der potenziellen Nebenläufigkeit mehrerer Kontextadaptionen und damit auch Kalibrierungen ergeben können?
- Wie kann für die Kalibrierung eine intuitive Beschreibung der momentanen respektive zu ändernden Situationsinterpretationen erreicht werden?
- Wie kann, aufbauend auf einer solchen Beschreibung, die Benutzerschnittstelle der Kalibrierung die Ubiquitätsbedingungen erfüllen?

4.1 Reprogrammierbarkeit eines Systems

Gemäß den Ausführungen in 3.3 kann das Frame-Problem durch eine vollständige Reprogrammierbarkeit der Kontextadaption kompensiert werden. Die Vollständigkeit ist dabei von wesentlicher Bedeutung, da das bloße Hinzufügen von "Wissen" (Regeln, Code, etc.) über die Wirklichkeit zwar einzelne Symptome (SUBs) des Frame-Problems unterdrücken kann, die unterschiedliche Interpretation (Divergenz zwischen Modell und Wahrnehmung der Wirklichkeit durch den Benutzer) aber nicht in allen Fällen beseitigt [78] und dadurch Folge-SUBs erzeugen kann.

Reprogrammierung darf Ubiquität nicht behindern.

Diese Forderung nach vollständiger Reprogrammierbarkeit erzeugt in ubiquitären Anwendungen aber zwei Probleme, die sich aus den beiden Ubiquitätsbedingungen der Nutzbarkeit und Verfügbarkeit in möglichst vielen Situationen ergeben.

4.1.1 Reprogrammierung zur Laufzeit

Die erste Bedingung der Nutzbarkeit einer ubiquitären Anwendung in möglichst vielen Situationen bedeutet, dass eine ubiquitäre Anwendung während ihrer Kalibrierung nutzbar bleiben sollte, die Reprogrammierung also zur Laufzeit erfolgt. Zwar ist die Kalibrierung und damit auch die Reprogrammierung zeitlich entkoppelt von der Nutzung der betroffenen Funktionen, weil die notwendigen Änderungen vor oder nach dem Auftreten eines SUBs erfolgen. Jedoch bestehen ubiquitäre Anwendungen typischerweise aus einer größeren Zahl von Funktionalitäten. Von der Kalibrierung nicht direkt betroffen Funktionen könnten also durchaus währenddessen nutzbar sein, sofern es sich beispielsweise um vollständig automatisierte Funktionen handelt, die im Hintergrund ablaufen.

Ein Beispiel ist die in 2.3 beschriebene Anwendung. Würde die Funktion der kontextabhängigen Nachrichtenverteilung für eine Kalibrierung deaktiviert, beispielsweise um eine Änderung am Quellcode vorzunehmen, wie etwa in [54] erforderlich, wären von dieser Abschaltung auch alle abhängigen Funktionen (siehe 2.2 - Einkaufsliste, Telefonsystem etc.) betroffen. Dadurch könnte ein Anruf während der Kalibrierung nicht mehr richtig signalisiert oder umgeleitet werden.

4.1.2 Reprogrammierung unterschiedlicher Realisierungen von Adaption

Die Kontextadaption selbst ist eine ubiquitäre Funktion.

Ein weiteres Problem der vollständigen Reprogrammierbarkeit ergibt sich aus der zweiten Ubiquitätsbedingung, also der Verfügbarkeit in möglichst vielen Situationen. Diese Bedingung zielt darauf ab, die Kontextadaption auch dafür einzusetzen, eine von externen Ressourcen abhängige Funktion in Situationen mit wechselnder Infrastruktur (verfügbaren HW/SW-Komponenten) zu realisieren. Dies schließt natürlich auch die Funktionalität der Kontextadaption selbst mit ein.

In der Regel benutzte der in 2.3 beschriebene Prototyp komplexe Entscheidungsregeln, die aus programmierten Scripts oder auch Markov Modellen bestanden. In bestimmten Situationen wie dem Urlaubsmodus, in denen nur eine sehr geringe Rechenleistung zur Verfügung stand, basierten die Entscheidungen dagegen auf einfachen boolschen Regeln, die als Makro in einen kleinen Mikrocontroller geladen wurden, damit der Server deaktiviert werden konnte.

Kalibrierung muss unabhängig von konkreten technischen Realisierungen der Kontextadaption funktionieren.

Daraus ergibt sich die Anforderung an einen generischen Mechanismus für die Reprogrammierung, der in der Lage ist, auch unterschiedliche technische Realisierungen der Kontextadaption auf einheitliche Weise zu verändern (eine Art von Metaregeln). Dabei muss auch gewährleistet sein, dass bei der Reprogrammierung ein vollständiger Übergang zwischen zwei technischen Realisierungen erzielt werden kann.

Für den in 2.3 beschriebenen Prototypen trat im Urlaubsmodus ein SUB auf, weil eine zufällige Auslösung des Bewegungsmelders zu einem Verlassen des Urlaubsmodus führte, obwohl sich niemand in der Wohnung befand. Mit den beschränkten Möglichkeiten der boolschen Schaltmakros des Mikrocontrollers ließ sich dieses SUB nicht kalibrieren. Aus diesem Grund musste die kontextabhängige Urlaubsende-Erkennung mithilfe eines komplexeren Regelmechanismus reprogrammiert werden. Damit die dafür notwendige Rechenleistung zur Verfügung stand, musste auch die Urlaubsadaption dahingehen kalibriert werden, einen Computer mit ausreichend Rechenleistung von der automatischen Abschaltung aller Geräte auszunehmen.

Das obige Beispiel zeigt sehr klar die Bedeutung dieser Anforderung. In ubiquitären Systemen bilden Funktionen und Realisierung nicht unbedingt eine Einheit. Eine konkrete Realisierung ist dagegen in der Regel abhängig von ei-

ner bestimmten Plattform oder zumindest deren technischen Möglichkeiten. Reprogrammierung zum Zwecke der Kalibrierung muss daher auf einer Ebene stattfinden, die von konkreten technischen Realisierungen abstrahiert. Zum Beispiel indem auf der Ebene von Funktionen oder sogar Anpassungsbedürfnissen agiert wird, unabhängig davon, ob sich dahinter einen Satz von Regeln, ein neuronales Netz oder ein Stück Programmcode verbirgt. Die technische Realisierung darf dabei höchstens als weitere Funktionsabhängigkeiten oder Bedürfnis bezüglich deren Leistungsfähigkeit (z.B. der Hardware) auftauchen.

4.1.3 Zusammenfassung und Diskussion der Anforderungen an die Reprogrammierbarkeit

Aus den beiden Ubiquitätsbedingungen (Nutzbarkeit und Verfügbarkeit in möglichst vielen Situationen) ergeben sich zwei wichtige Anforderungen an die vollständige Reprogrammierbarkeit der Kontextadaption (als Bestandteile der Kalibrierung ubiquitärer Anwendungen). Die Kalibrierung entspricht der Kompensation des Frame-Problems in nichtubiquitären Systemen durch die Spontane Hülle (manuelle Mediatorfunktion des Benutzers). Im Unterschied zu dieser besteht bei der Kalibrierung aber keine zeitnahe Verknüpfung zwischen den für die Kompensation notwendigen Festlegungen und der Nutzung einer Funktion. Statt dessen erfolgt die Kalibrierung vor dem Auftreten eines (absehbaren) SUB oder danach und kollidiert daher nicht mit der Ubiquitätsbedingung der Nutzbarkeit, sofern die Kalibrierung zur Laufzeit erfolgen kann und keine anderen Funktionen stört. Das Analogon der verhaltensverändernden Bestandteile der Benutzerschnittstelle in der Spontanen Hülle ist bei der Kalibrierung die Reprogrammierbarkeit. Aufgrund der zweiten Ubiquitätsbedingung muss die Reprogrammierbarkeit aber eine Möglichkeit bieten, verschiedene technische Realisierungsverfahren der Kontextadaption in einheitlicher Weise zu manipulieren. Dabei muss auch gewährleistet sein, dass die Kalibrierung eine Migration des Reprogrammierungsverfahrens von einem auf ein anderes technisches Verfahren erlaubt. Es ist also nicht damit getan, einen einzigen Regelmechanismus zu entwerfen. Dieser könnte mit bestimmten Aufgaben überfordert oder in bestimmten Hardwaresituationen nicht einsetzbar sein.

4.2 Nebenläufigkeit der Adaption

Ubiquitäre Anwendungen können offensichtlich aus mehreren gleichzeitig ausführbaren Funktionen bestehen (als Beispiel siehe die Fallstudie in 2.2).

Die Realisierung der Funktionen kann wiederum situationsabhängig (z.B. von lokalen Ressourcen) sein. Einige dieser Realisierungen können ihrerseits eine eigenständige ubiquitäre Anwendung darstellen (siehe auch nachfolgendes Beispiel). Folglich können in ubiquitären Anwendungen mehrere gleichzeitig ablaufende Kontextadaptionen miteinander durch gemeinsam genutzte physikalische (Geräte etc.) oder virtuelle (Kommunikationskanäle, gemeinsame Daten etc.) Ressourcen verknüpft sein.

UC Anwendungen können nebenläufige Adaptionen enthalten.

Innerhalb des in 2.3 beschriebenen Prototyps existiert eine kontextabhängige Nachrichtenfunktion. Diese entscheidet darüber, ob eine Benachrichtigung (Telefon, Email, etc.) zwischengespeichert, oder an den jeweiligen Standort des Benutzers weitergeleitet und auf welchem Gerät und in welcher Form (Sprache, Text, usw.) sie dort ausgegeben wird. Ein Teil dieser Ausgabegeräte ist wiederum kontextadaptiv (z.B. verschiedene Energiesparmodi oder Netzwerktechnologien je nach geforderter Leistung). Die Kontextadaptionen der Nachrichtenvermittlung und die der Endgeräte sind über die gemeinsam genutzte Geräteressource und Kommunikationskanäle miteinander verknüpft. Ähnliches gilt für andere kontextadaptive Funktionen (Einkaufsliste etc.), welche den Nachrichtendienst benutzen und mit diesem über gemeinsame Daten (Nachricht) verknüpft sind.

Diese gegenseitige Abhängigkeit erfordert sowohl eine zeitliche als auch inhaltliche Abstimmung (im Folgenden kurz Synchronisation) von zwei oder mehr miteinander verknüpften nebenläufigen Kontextadaptionen. Zwar können sich die betroffenen Funktionen selbstständig an unterschiedliche Situationen anpassen, jedoch ist dadurch keinesfalls zwangsläufig eine unterbrechungsfreie Nutzbarkeit der miteinander verknüpften Funktionen gegeben.

Nebenläufige Adaptionen können sich gegenseitig beeinflussen.

Am einfachsten erschließt sich die Problematik bei der Betrachtung einer aktiven Kommunikationsbeziehung zwischen dem kontextadaptiven Nachrichtendienst und einem kontextadaptiven Endgerät. Das Gerät kann aufgrund bestimmter Gesichtspunkte und Leistungsmerkmale (Nähe zum Empfänger etc.) für eine bestimmte Nutzungssituation ausgewählt werden. Während der Nachrichtenübertragung, bei der es sich auch um ein Live-Video handeln könnte, kann eine Situation entstehen, die zu einer Adaption des Endgerätes (Netzwerkverbindung wird schlecht, Batterie wird schwach) und damit zu einer Unterbrechung (beispielsweise durch Wechsel von WLAN nach GPRS) führt.

Aufgrund der Kontextadaption ist der Nachrichtendienst natürlich in der Lage, diese Unterbrechung zu erkennen und gegebenenfalls ein anderes Gerät oder auch das ursprüngliche unter anderen Leistungsbedingungen (Bandbreite etc.) erneut zu verbinden.

Dabei entsteht aber eine Unterbrechung der Nutzung, die durch eine koordinierte Abstimmung zwischen Nachrichtendienst und Endgerät hätte vermieden werden können. Zum Beispiel hätte das Endgerät seine Adaption verzögern können, wenn die Nachricht bereits fast fertig übertragen gewesen wäre. Oder das Gerät hätte dem Nachrichtendienst die Art der bevorstehenden Adaption mitteilen können, damit dieser eine schnelle Wiederaufnahme der Verbindung vorbereiten hätte können.

Bei der Synchronisation von nebenläufigen Kontextadaptionen sind daher einige wichtige Probleme zu lösen. Da auch die Kalibrierung selbst eine Art der Adaption der Kontextadaption darstellt, übertragen sich diese Probleme auch auf die Kalibrierung nebenläufiger Kontextadaptionen und müssen durch entsprechende Anforderungen berücksichtigt werden. Dies ist besonders dann der Fall, wenn ein ubiquitäres System wie im Fall der Smart Spaces gleichzeitig von mehr als einer Person gleichberechtigt genutzt werden kann. Besonders bei unbewusster Kalibrierung können leicht Situationen entstehen, in denen zwei konkurrierende Kalibrierungen gegeneinander aufgelöst werden müssen.

Ein gutes Beispiel für unbewusste Kalibrierung war das automatische Umschalten zwischen den manuellen Kalibrierungen unterschiedlicher Benutzer. Wird ein ubiquitäres System aber von mehr als einer Person gleichzeitig genutzt, müssen die beiden dazugehörigen Kalibrierungen miteinander abgeglichen werden, da sie ja durchaus das System in zwei gegensätzliche Richtungen hin verändern könnten. Handelt es sich um nicht gleichberechtigte Nutzer, wie beispielsweise in einem Auto, können entstehende Kalibrierungskonflikte noch relativ leicht gelöst werden. In Anwendungen wie dem in der Fallstudie beschriebenen intelligenten Wohnraum, kann dieser Abgleich allerdings sehr schnell sehr kompliziert werden, beispielsweise weil manche Kalibrierungen in ihrer Wirkung auf einen einzigen Raum beschränkt sind, die Benutzer sich aber in unterschiedlichen Räumen aufhalten, dann aber wieder in einem gemeinsamen Raum zusammenkommen können.

4.2.1 Adaptionspropagation

Um eine einfachere Darstellung der Propagationsproblematik zu erreichen, wird zunächst ohne Beschränkung der Allgemeinheit angenommen, dass sich die im letzten Abschnitt beschriebenen Verknüpfungen zwischen nebenläufigen Kontextadaptionen als lineare Abhängigkeiten (Schichtung) darstellen lassen. Weiterhin, dass die Adaption in jeder Schicht ausschließlich auf Informationen basiert, die entweder aus direkt benachbarten Schichten stammen, oder in der betreffenden Schicht selbst erzeugt wurden. Dies bedeutet, dass Kontextadaptionen in nicht direkt benachbarten Schichten vollständig voneinander isoliert sind. Ein Informationsaustausch mit diesen Schichten ist lediglich durch indirekte Propagation (Weiterreichen von Schicht zu Schicht) möglich. In der Realität gelten diese Einschränkungen beispielsweise für miteinander kommunizierende Protokoll-Stacks (ISO/OSI Modell). Kompliziertere Abhängigkeiten lassen sich aber zum Zwecke der Synchronisation von Kontextadaption in mehrere solcher Schichtungen aufspalten.

Das *Adaptation Propagation Problem* (APP) entsteht aufgrund einer gewissen Verzögerung zwischen der Erkennung einer Situation (in Schicht B) aus den im Datenstrom zweier benachbarter Schichten A und B enthaltenen Merkmalen einer Situation (Kontext) und der Erkennung (in B) der Auswirkungen einer sich eventuell daraus ergebenden Adaption (von A) auf den Datenstrom.

Zum Zeitpunkt t_1 kann in Schicht A eine Verringerung der Signalqualität festgestellt werden. Die Kontextadaption in Schicht A kommt deshalb zu der Entscheidung, auf eine andere Übertragungstechnik zu wechseln. Diese Adaption ist zum Zeitpunkt $t_2>t_1$ abgeschlossen. Des Weiteren kann in der von A abhängigen Schicht B zum Zeitpunkt $t_3>=t_1$ eine Verringerung der Nettobandbreite wahrgenommen werden. Das erneute Ansteigen der Nettobandbreite aufgrund der Adaption in Schicht A kann dagegen erst zum Zeitpunkt $t_4>t_3$ mit einer Verzögerung von (t_2-t_1) wahrgenommen werden.

Signalisierungsverzögerungen zwischen nebenläufigen Adaptionen können zu Kollisionen führen.

Eine solche Verzögerung *APD* (*Adaptation Propagation Delay*) ergibt sich immer dann, wenn die Kontextadaption nicht in Echtzeit erfolgen kann. Sie kann jedoch auch entstehen, wenn für die Erkennung einer Situation auf unterschiedlichen Schichten unterschiedliche Merkmale (Kontext) Verwendung finden. Folglich reicht auch eine einfache Signalisierung von auf niedrigeren Schichten begonnenen Adaptionen nicht aus, um dem Propagationsproblem zu begegnen.

Kommen in den Schichten A und B unterschiedliche Merkmale für eine Erkennung eines Absinken der Nettobandbreite zum Einsatz (z.B. unterschiedliche Toleranzgrenzen) kann der Erkennungszeitpunkt in B vor dem in A liegen ($t_3<t_1$). Auch wenn die Kontextadaption in A in Echtzeit erfolgt ($t_2=t_1$), kann die Auswirkung der Adaption von A in B lediglich mit einem *APD* von (t_3-t_1) wahrgenommen werden.

Aus dem *APD* können sich verschiedene Typen von negativen Kollisions- und Interaktionseffekten zwischen nebenläufigen Kontextadaptionen in verschiedenen Schichten ergeben.

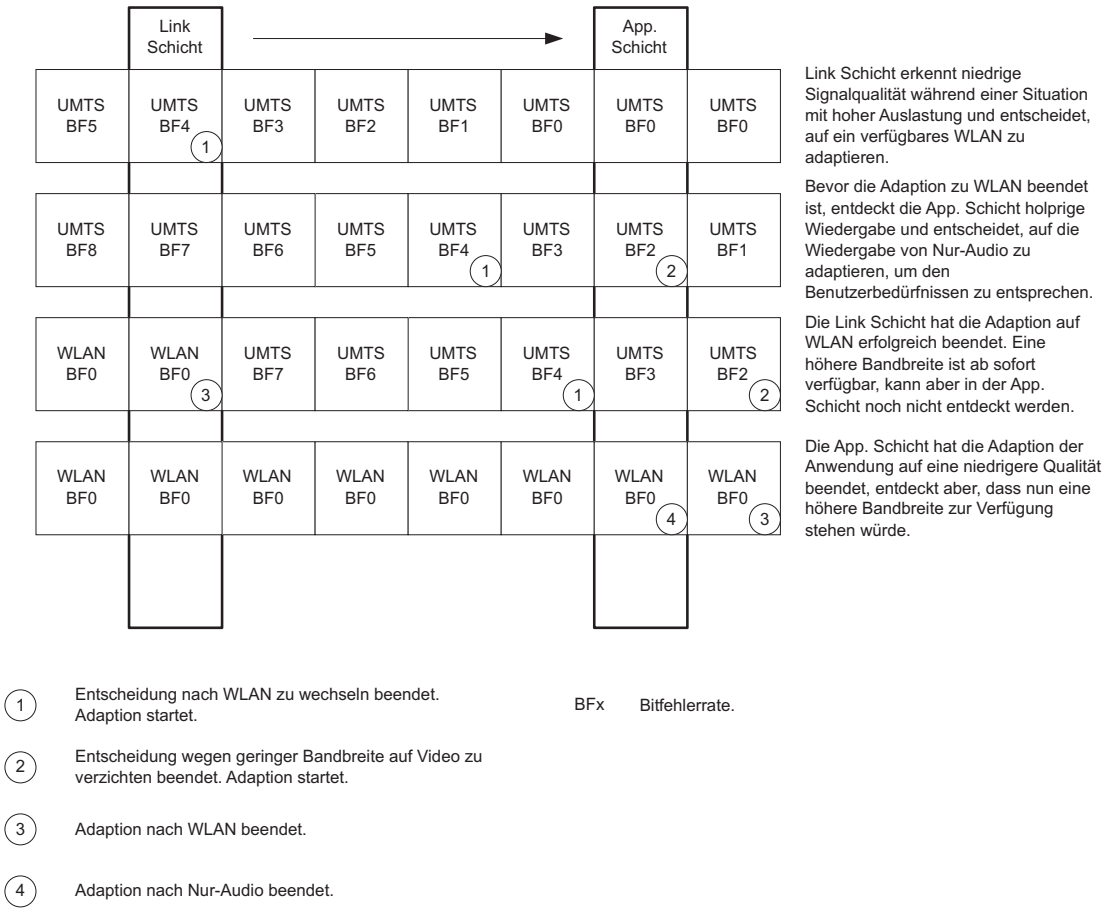


Abb. 14: Das Adaptation Propagation Problem (APP)

Abb. 14 zeigt ein Beispielszenario für einen solchen APP Effekt. Aufgrund des unterschiedlichen Kontextes für die Erkennung der gleichen Situation in zwei nebenläufigen Adaptionen kommt es zu einer suboptimalen Adaption des Gesamtsystems (geringere Qualität bei mehr Bandbreite). Im weiteren Verlauf könnte es auch zu einem Resonanzeffekt (eine Adaption bedingt eine

andere und umgekehrt) kommen, wenn die niedrigere Schicht aufgrund der geringeren Bandbreitenauslastung wieder auf eine niedrigere Bandbreite adaptieren würde, während die Applikationsschicht gleichzeitig versucht, die höhere Bandbreite wieder sinnvoll zu nutzen.

4.2.2 Propagationsunterbrechung

Verzögerungen bei der Signalisierung von Adaptionen nebenläufiger Kontextadaptionen und daraus entstehende APP Effekte können durch den Umstand vergrößert werden, dass zwischen den Schichten der gekoppelten Konfigurationen noch weitere Schichten liegen können, die ihrerseits von Adaption betroffen sein können (siehe Abb. 15).

Kalibrierungen können Synchronisation anderer Adaptionen unterbrechen.

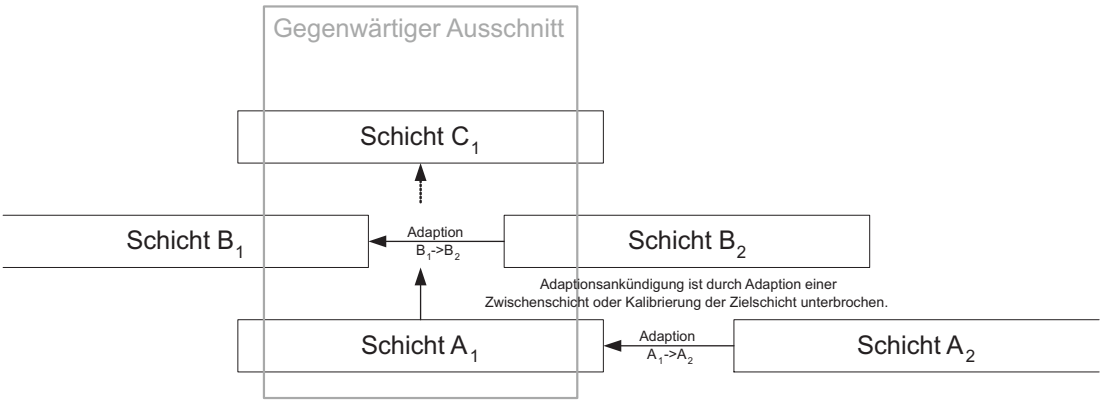


Abb. 15: Das Adaptation Propagation Interruption Problem (APIP)

Die Informationen einer Adaptionssignalisierung können dabei schlimmstenfalls verloren gehen. Dies lässt sich zwar durch geeignete Puffermechanismen verhindern, führt aber zu einer zusätzlichen Verzögerung. Ein Sonderfall dieses Adaptation Propagation Interruption Problems (APIP) kann bei der Kalibrierung auftreten, wenn die zu synchronisierende Kontextadaption gerade selbst das Ziel einer Adaption ist, also in genau dem Moment möglicherweise vollständig reprogrammiert wird, während eine gekoppelte Kontextadaption gerade den Versuch unternimmt, den Synchronisationsprozess zu starten.

Die nebenläufigen Kontextadaptionen zweier Schichten (A, B) sind miteinander gekoppelt. Nach dem Auftreten eines SUB-Phänomens wird eine Kalibrierung der Kontextadaption auf Schicht B vorgenommen. Die Kalibrierung ist ebenfalls eine Adaption von Schicht B (genauer gesagt der Kontextadaption auf Schicht B). Aufgrund des APP kann die Signalisierung dieser Adaption an A verzögert sein.

Innerhalb dieser Verzögerung könnte die Kontextadaption in A ihrerseits eine Adaption starten, die mit B synchronisiert werden müsste. Wegen der Kalibrierung steht dort aber kein geeigneter Synchronisierungspartner mehr zur Verfügung. Bereits abgeschickte Synchronisierungsinformationen sind möglicherweise verloren. Erneute Versuche und weitere Verzögerungen bei der Signalisierung von A nach B mit APP Effekten sind die Folge.

4.2.3 Mehrfachkopplungen

Ein weiteres Problem für die Synchronisierung von nebenläufigen Kontextadaptionen sind Mehrfachkopplungen, das heißt Verknüpfungen und Abhängigkeiten zwischen mehr als zwei Adaptionen.

An einer Telefonkonferenz sind die drei Endgeräte A, B, C beteiligt. Die für A zuständige Kontextadaption stellt fest, dass eine Möglichkeit bestünde, auf ein Netzwerk höherer Bandbreite zu wechseln und damit von einer Audio auf eine Videokonferenz zu adaptieren. Bei der Koordination mit den Kontextadaptionen der anderen Geräte stellt sich heraus, dass B in der Lage wäre, der Adaption zu folgen, C jedoch nicht.

Mehrfachkopplungen nebenläufiger Adaptionen können Ubiquitätsbedingungen widersprechen.

Normalerweise müsste in solchen Fällen die Adaption unterbunden werden. Allerdings geht mit einer solchen Koordinationsstrategie auch eine Einschränkung der Ubiquität für eine Mehrheit der Beteiligten einher. Dieses Least Common Denominator Problem (LCDP) für gekoppelte Adaptionen wird mit zunehmender Zahl der gekoppelten Adaptionen schwerwiegender.

Eine größere Anzahl an mobilen Geräten formt ein drahtloses ad hoc p2p Netz für einen gemeinsamen Datenaustausch (Gruppenarbeit, Spieler in einem Zug, Autos in einem Stau etc.). Zu einem späteren Zeitpunkt entfernen sich zwei Geräte gemeinsam aus dem Bereich des ad hoc Netzwerkes. Um die Verbindung aufrecht zu erhalten, müssten alle Geräte auf den kleinsten gemeinsamen (und teuersten) Nenner, nämlich eine Mobilfunkverbindung adaptieren.

Die günstigste Lösung bestünde aber darin, wenn lediglich zwei Geräte mit der Möglichkeit, zwei Verbindungen gleichzeitig zu bedienen, eine Brücke zwischen den ad hoc Netzen der beiden Gruppen herstellen und die Kosten auf alle anderen verteilen würden.

Aus dem LCDP ergibt sich eine Anforderung, bei der Synchronisierung von Kontextadaption im Allgemeinen und bei deren Kalibrierung im Besonderen eine Möglichkeit zu haben, Struktur und Art der Kopplung nebenläufiger Kontextadaptionen zu erkennen und aktiv zu beeinflussen.

4.3 Beschreibung von Adaptionsverhalten

Nach 3.3 besteht die Kalibrierung der Kontextadaption zur Vermeidung des Frame-Problems aus einer doppelten Vermeidungsstrategie. Um SUB-Phänomene vorhersehen zu können, muss ein Benutzer die in einer ubiquitären Anwendung gerade gültigen Situationsinterpretationen möglichst genau kennen. Dies kann in Form von Handbüchern, Online-Hilfen oder erläuternden Kommentaren der Anwendung geschehen und erlaubt ihm eine Abschätzung, wie sich ein System in einer (für den Benutzer) vorhersehbaren neuen oder Ausnahmesituation verhalten wird. Zeichnet sich ein unerwünschtes Verhalten ab, bieten sich dem Benutzer die folgenden Optionen:

- Sofern es in seiner Macht steht, kann der Benutzer das Eintreten einer absehbar kritischen Situation durch Ändern des eigenen Verhaltens verhindern.
- Der Benutzer kann die Wahrnehmung des Systems dahingehend täuschen, dass eine Situation erkannt wird, die zwar nicht der Wirklichkeit entspricht, aber ein Verhalten produziert, das den tatsächlichen Benutzerbedürfnissen entspricht. Im Beispiel des Kühlschrank SUB aus 1.2 könnte der Benutzer etwa eine Registrierung der Partyvorräte durch Abziehen der RFID-Etiketten oder Abdecken des Barcodescanners unterbinden.
- In der Regel wird der Benutzer aber die Situationsinterpretation des Systems verändern (reprogrammieren). Im einfachsten Fall kann das auch ein vorübergehendes Deaktivieren einer Funktion oder des ganzen Systems sein.

Die letzte Möglichkeit setzt wiederum ein geeignetes Beschreibungsmittel voraus, das es dem Benutzer erlaubt, die Reprogrammierung der Situationsinterpretationen eines Systems vorzunehmen.

Spezifikation des Adaptionsverhaltens muss für Benutzer verständlich sein.

Das Problem besteht nun generell darin, eine intuitiv verständliche und auch intuitiv verwendbare Beschreibung zu konstruieren, die es auch technisch weniger versierten Anwendern erlaubt, das Interpretationsverhalten eines Systems zu verstehen und zu verändern. Die besondere Schwierigkeit besteht in der in 4.1 geforderten möglichst vollständigen Reprogrammierbarkeit. Das

Lesen und inkrementelle Modifizieren einiger natürlichsprachiger Regeln ist beispielsweise kein geeigneter Mechanismus, da durch lediglich neues Wissen hinzugefügt, aber eine grundlegende Divergenz in der Interpretation der Wirklichkeit nicht beseitigt wird [78].

Glücklicherweise beschränkt sich die Forderung nach vollständiger Reprogrammierbarkeit lediglich auf das Verhalten der Situationsinterpretation (die Kontextadaption) und nicht auf ein beliebiges Anwendungsverhalten. Der Benutzer muss also nicht in der Lage sein, beliebige Anwendungen intuitiv zu programmieren, sondern lediglich ein beliebiges Adaptionsverhalten auf der Basis von vorhandenen Änderungsmöglichkeiten. Dennoch bleibt die intuitive Verständlichkeit und Verwendbarkeit eine große Herausforderung. Die wichtigsten Anforderungen bezüglich der Intuitivität sind im Folgenden kurz zusammengefasst.

4.3.1 Vollständigkeit und Multiaspekt-Darstellung

Ein wichtiges Merkmal intuitiver Darstellungen ist, dass sich die wesentlichen Informationen mit einem "Blick" erfassen lassen. Im Gegensatz dazu basieren die gängigen Beschreibungssprachen der Softwaretechnik meist auf mehreren spezialisierten Darstellungen einzelner Aspekte (Sichtweisen), die jeweils einzeln betrachtet werden müssen, aber erst zusammengenommen ein vollständiges Bild ergeben.

Analytische Darstellungen aus verschiedenen Blickwinkeln sind für Benutzer ungeeignet

Diesen Unterschied kann man leicht in einem eigenen Experiment nachvollziehen. Dazu werfe man einen sehr kurzen (<1s) Blick auf einen beliebigen Gegenstand. Trotz der kurzen Betrachtungszeit hat man in der Regel einen vollständigen Eindruck des Gegenstandes gewonnen und sich bereits eine Meinung darüber gebildet, die aus sehr vielen Aspekten einer einzigen Darstellung gewonnen wird. Dieser erste Eindruck ändert sich auch bei längerer Betrachtungsweise nur selten (Primacy Effekt [82]). Statt dessen werden weitere Details gesucht, welche den ersten Eindruck stützen. Details, die nicht dem ersten Eindruck entsprechen, werden dagegen meist unterbewertet. Nur wenn der erste intuitive Eindruck ein ausreichendes Interesse geweckt hat, wird möglicherweise in eine Analysephase eingetreten. Erst in dieser Phase werden dann genauere Beobachtungen aus verschiedenen Perspektiven in Betracht gezogen.

Für eine intuitive Beschreibung von Situationsinterpretationen eines kontextadaptiven Systems ergeben sich daraus die folgenden Anforderungen:

- Die Beschreibung sollte möglichst viele Aspekte in einer einzigen Sichtweise vereinen.
- Die Informationsdichte sollte möglichst hoch sein.

4.3.2 Übersichtlichkeit und Gegenständlichkeit

Zu hohe Informationsdichten können für die intuitive Verständlichkeit einer Darstellung aber auch schädlich sein. Ein einfaches Beispiel dafür ist die Seite eines Buches, deren Inhalt und Bedeutung sich in der Regel nicht auf einen Blick erschließt. Ein anderes Beispiel sind komplexe elektronische Schaltpläne aber auch komplexere UML-Diagramme, wie sie in Kapitel 6 für die Spezifikation eines Framework der Kalibrierung verwendet werden.

Innerhalb der menschlichen Wahrnehmung der noch komplexeren Wirklichkeit existieren aber verschiedene Verfahren, Übersichtlichkeit in detailreichen Darstellungen zu erzeugen. So wird eine optische Wahrnehmung zunächst in Bereiche ähnlicher Eigenschaften unterteilt (Gegenstände). Die erkennbaren Details eines solchen Objektes hängen zudem von der Beobachtungsintensität ab. Wichtige Details (Form, Größe, Geschwindigkeit, Lärm) können am schnellsten wahrgenommen werden. Weniger wichtige Details (Farbe, Temperatur, Oberflächenbeschaffenheit etc.) lassen sich nur bei genauerer Beobachtung oder Untersuchung erkennen. Erreicht wird dieser Effekt durch entsprechende Konstruktion des sensorischen Apparates. So kann das menschliche Auge beispielsweise hochauflösende und farbige Wahrnehmung nur in einem sehr kleinen Ausschnitt des Blickfeldes durchführen, der vergleichbar einer Lupe auf die interessanten Stellen eines Objektes fokussiert werden muss. Bewegungen lassen sich dagegen über das gesamte Blickfeld hinweg wahrnehmen. Andere Sensoren wie beispielsweise der Tastsinn sind stark in ihrer Reichweite begrenzt, erfordern also eine physische Annäherung an das zu untersuchende Objekt.

Übersichtlichkeit einer Beschreibung lässt sich durch Berücksichtigung dieser Wahrnehmungsfaktoren erzielen (Schlagzeilen, blinkende Warnlampen, Strukturierung usw.). Daraus ergeben sich die folgenden Anforderungen an eine intuitive Beschreibung der Kalibrierungsinhalte:

- Die Anzahl der erfassbaren Details sollte von der Beobachtungsintensität abhängen. Die wichtigsten Informationen sollten auch am schnellsten (und offensichtlichsten) erkennbar sein.
- Die Beschreibung sollte objektorientiert sein. Ein Bezug dieser Objekte zu realen Gegenständen in der Umgebung erleichtert dabei das Auffinden interessanter Stellen innerhalb einer Beschreibung und das erkennen weiterer Details.

4.4 Benutzerschnittstellen für Kalibrierung

Das letzte große Problem der Kalibrierung ist die Einbettung der intuitiven Beschreibung in eine Benutzerschnittstelle. Diese sollte selbst so weit wie möglich ubiquitär, das heißt in unterschiedlichen Situationen verfügbar und nutzbar sein.

Allerdings handelt es sich dabei um keine kritische Anforderung. Die Kalibrierung kann sehr wohl auch in nichtubiquitärer Weise erfolgen, das heißt von vordefinierten Geräten oder Stellen aus und unter Voraussetzung einer exklusiven Interaktion mit dem Benutzer. Allerdings verringern sich dadurch die Chancen, SUB-Phänomene vor ihrem ersten Auftreten oder eine Wiederholung derselben zu verhindern. Auf der anderen Seite kann es aber aus Sicherheitsgründen durchaus sinnvoll sein, die Kalibrierung etwa auf ein bestimmtes Gerät oder einen bestimmten Ort zu beschränken.

Ist trotzdem eine ubiquitäre Kalibrierung erwünscht, existieren bereits eine Reihe von möglichen Arten der Realisierung einer solchen Schnittstelle von der Augmented Reality bis hin zu multimodalen Schnittstellen, die gleich eine Reihe von unterschiedlichen Interaktionsformen (Sprache, Gesten etc.) miteinander vereinen. Im Sinne der Ubiquität gibt es aber keine besonders geeignete Form der Realisierung. Statt dessen muss aufgrund der Situation von Umgebung und Benutzer eine jeweils geeignete Form der Interaktion gewählt werden.

Für diese Arbeit ergibt sich daraus die Anforderung, dass die Benutzerschnittstelle der Kalibrierung von der Art ihrer Realisierung abstrahieren muss, eine Umsetzung also sowohl in Sprache, grafischer Darstellung auf einem Terminal oder anderen Formen möglich ist und zwischen diesen Möglichkeiten im laufenden Betrieb adaptiert werden kann.

Teil II

Modellierung, Methodik und Framework für kalibrierbare Kontextadaption

KAPITEL 5 Modellierung

Wie sieht ein formales Modell der Kontextadaption aus?
Wie wird dessen Kalibrierbarkeit definiert?

In 3.3 war als Kompensation des Frame-Problems in ubiquitären Anwendungen das Konzept der Kalibrierung vorgestellt worden. Diese basiert auf der Idee der K-Modelle. Ein solches enthält alle FP-kritischen Bestandteile einer ubiquitären Anwendung. Dabei handelt es sich um alle Funktionen, die mit der Erkennung und Interpretation von Situationen eines Systems zu tun haben. Diese Funktionen waren in 3.2 unter dem Begriff der Kontextadaption zusammengefasst worden. Ein K-Modell ist folglich die explizite Modellierung der kalibrierbaren Kontextadaption eines ubiquitären Anwendungssystems.

Die Kalibrierung wiederum ist ein Kompensationsmechanismus für das Frame-Problem, der auf einer möglichst vollständigen Reprogrammierbarkeit der durch das K-Modell beschriebenen FP-kritischen Bestandteile eines Systems basiert. Dazu wird das durch ein K-Modell definierte Adaptionsverhalten des ubiquitären Systems auch in der Realisierung explizit vom Rest der Anwendung abgespalten. Das so entstehende Kontextadaptions-Subsystem wird mit der Fähigkeit versehen, über eine eigene Benutzerschnittstelle unter Verwendung auch für Endbenutzer geeigneter Beschreibungsmittel möglichst vollständig reprogrammiert werden zu können. Schnittstelle, Beschreibungssprache, K-Modell und Reprogrammierbarkeit bilden somit zusammen den Kalibrierungsmechanismus. Das folgende Kapitel beschreibt eine Formalisierung der K-Modelle und ihrer Reprogrammierbarkeit. Benutzerschnittstelle und Beschreibungstechnik werden im darauf folgenden Kapitel zusammen mit einem Framework für kalibrierbare Kontextadaption beschrieben.

5.1 Überlegungen zur Modellbildung

Bevor in den nächsten Abschnitten ein formales K-Modell diskutiert wird, sollen an dieser Stelle zunächst die der Modellierung zugrunde liegenden Entscheidungen und Konzepte diskutiert werden.

5.1.1 Kalibrierung

Wie am Anfang des Kapitels bereits zusammengefasst, handelt es sich bei der Kalibrierung um einen Kompensationsmechanismus für Abweichungen zwischen einer in einem Anwendungssystem enthaltenen Modellierung der Realität zum Zeitpunkt der Entwicklung und der möglicherweise veränderten Wirklichkeit, in der sich ein Benutzer zum Zeitpunkt der Nutzung befindet (siehe Kapitel 3).

Das Frame-Problem besagt, dass solche Modelldivergenzen prinzipieller Natur sind und durch die Art der Modellierung (als Symbolsystem mit extrinsischen Bedeutungen) an sich und eine veränderliche (nicht vorhersehbare) Wirklichkeit verursacht werden. Einfach ausgedrückt, Modelle können "veralten" beziehungsweise sich unterscheidende Modelle längere Zeit für identisch gehalten werden. Aufgrund des Frame-Problems besteht im Allgemeinen also keine Möglichkeit, die Entstehung der Divergenzen von vornherein zu verhindern. Weiterhin ist es auch im Allgemeinen nicht möglich, ein technisches System zu konstruieren, das solche Divergenzen selbstständig erkennt und beseitigt. Solche Thesen über Ursache und Wirkung sind natürlich Gegenstand teils heftiger wissenschaftlicher aber auch philosophischer Diskussionen (siehe [83]). Für die vorliegende Arbeit ist es dagegen völlig ausreichend, aus der praktischen Beobachtung der Symptome auf die Existenz des Problems zu schließen. Genauso wie aus der Tatsache, dass bisher keine allgemeine und mit vertretbarem Aufwand realisierbare Lösung gefunden wurde (wie auch in [78] dargelegt) und dass auch Menschen von dem Problem betroffen sind, angenommen werden kann, dass eine allgemeine Lösbarkeit, besonders unter Einbeziehung der Erkennung von Benutzerbedürfnissen, nahezu ausgeschlossen werden kann.

Statt einer möglicherweise gar nicht existierenden Lösung wird daher lediglich nach einer Kompensationsmöglichkeit gesucht, die einen akzeptablen Umgang mit der Problematik erlaubt. Für die Kompensation stehen dabei zwei prinzipielle Möglichkeiten zur Verfügung. Im ersten Ansatz verändert der Benutzer die Wirklichkeit, respektive die vom System wahrnehmbare Wirklichkeit oder das gelieferte Ergebnis so, dass die Auswirkungen der Modelldivergenz im Ergebnis für den Beobachter neutralisiert wird. Eine andere

Alternative ist, das im System enthaltene Modell durch den Benutzer von außen so zu verändern, dass die Divergenz beseitigt wird, solange bis eine neue Divergenz entsteht.

Die erste Kompensationsmöglichkeit ist in 3.2.4 als Spontane Hülle bezeichnet worden. Diese definiert auf der Basis der Benutzerschnittstelle ein entsprechendes Kompensationsverhalten des Nutzers (Mediatorfunktion bzw. manuelle Adaption) und funktioniert für die meisten gängigen Softwareanwendungen ausreichend gut. Aufgrund der vielen direkten und bewussten Interaktionen zwischen Nutzer und System, eignet sich dieses Verfahren aber nicht für ubiquitäre Anwendungen. Diese sollten nämlich auch in Situationen nutzbar sein, in denen die Interaktionsfähigkeiten eines Anwenders mehr oder weniger stark eingeschränkt sind.

Die zweite Kompensationsmöglichkeit ist die Kalibrierung als das Gegenstück zur Spontanen Hülle in ubiquitären Anwendungen und basiert auf der zur Nutzung zeitversetzten Veränderung des in einem System enthaltenen Modells der Wirklichkeit. In 3.3 war dargelegt worden, dass dies auf eine möglichst vollständige Reprogrammierung der Anwendung zur Laufzeit hinausläuft. Das Ändern einiger Parameter oder Regeln, also das bloße Hinzufügen weiterer Informationen ist dagegen nicht dazu geeignet, das Frame-Problem und seine Symptome in Gestalt von SUB-Phänomenen zu kompensieren. Natürlich ist die völlige Reprogrammierung einer Anwendung durch den Benutzer zur Laufzeit ein unrealistisches Ziel. Statt dessen werden für die Kalibrierung nur die FP-kritischen Teile eines Systemverhaltens vom Rest der Anwendung isoliert und mit der Möglichkeit versehen, vom Benutzer zur Laufzeit reprogrammiert zu werden.

Einfach gesprochen teilt man ein ubiquitäres System sowohl konzeptionell als auch in der Realisierung in zwei Subsysteme. Das eine enthält die ubiquitär zu nutzenden Funktionen, das andere die FP-kritischen Kontextadaptionen, welche diese Ubiquität erzeugen. Letzteres Teilsystem wird mit einer eigenen Benutzerschnittstelle (tertiäre Interaktion, siehe auch Abb. 13 auf S. 72) ausgestattet, die unabhängig von den ubiquitären Funktionen ist und eine Veränderung des Adaptionsverhaltens erlaubt. Dadurch kann die Kompensation des Frame-Problems im Gegensatz zum Fall der Spontanen Hülle auch asynchron erfolgen, nämlich dann, wenn gerade genügend Interaktionsressourcen bereit stehen. Kalibrierung ist also genau genommen eine Spontane Hülle (manuelle Adaption) für Kontextadaption, während die Kontextadaption (automatische Adaption) die Spontane Hülle der ubiquitär zu nutzenden Anwendungsfunktionen (teilweise) ersetzt, beziehungsweise automatisiert um die Ubiquitätsbedingungen zu erfüllen.

Wegen dieser rekursiven Beziehung kann man die Kalibrierung auch wiederum als eine spezielle Art von Kontextadaption betrachten. In der vollautomatischen Variante erhöht das aber höchstens den Anwendungskomfort (von vielen kleineren zu wenigen größeren Strukturen). Ein sinnvolles Beispiel ist etwa ein schnelles Hin- und Herschalten zwischen den unterschiedlichen Kalibrierungen zweier Benutzer. Das Frame-Problem wird dabei aber nur von Ebene zu Ebene durchgereicht, so lange, bis eine echte manuelle Kalibrierung (manuelle Adaption) durchgeführt wird.

Herkömmliche direkte Benutzerschnittstellen können durch Kontextadaption nachgebildet werden

Allerdings kann die Kontextadaption eine manuelle Adaption auch emulieren, nämlich dann, wenn zwischen Ein-/Ausgabegeräten und System die Kontextinformationen einfach wie bei einer manuellen Benutzerschnittstelle ohne weitere Interpretation hindurchgereicht werden, aber originär von einem menschlichen Benutzer stammen.

Definition 10: *Emulationseigenschaft* der Kontextadaption

Direkte und bewusste Interaktionen zwischen Benutzer und System (Benutzerschnittstellen) können bei Bedarf als Kontextadaption in Form von manuellen Adaptionen (*Contextual Sensing* [46], d.h. Zuführen eines Kontextes zu seiner Verwendung ohne dazwischenliegende Interpretation) dargestellt werden.

Kalibrierung ist eine Adaption der Adaption

Dies erleichtert eine formale Definition und Realisierung der Kalibrierung erheblich. Sie kann dadurch einfach als Spezialfall der allgemeinen Kontextadaption beschrieben werden.

5.1.2 Kontextadaption

Kontextadaption, laut Definition eine explizite Anpassung des beobachtbaren Verhaltens oder des inneren Zustandes eines Systems an seinen Kontext (siehe Definition 6 in 3.2), kann zunächst als ein Prozess betrachtet werden.

Definition 11: *Prozess*

Prozesse sind eine gängige und intuitiv verständliche Betrachtungsweise von Vorgängen, die einen definierten Anfang und (mit Ausnahme von unendlichen Prozessen) auch ein Ende haben und ein Ergebnis produzieren.

Die Betrachtung als Prozess kommt der späteren Definition einer intuitiven Beschreibungssprache für Kontextadaption entgegen. Der Prozess der Kontextadaption involviert dabei typischerweise die folgenden Schritte [59] :

- Sammeln von Informationen aus der Umgebung über die gegenwärtige Situation in Form eines Kontextes.
- Erkennung einer bestimmten Situation aufgrund eines Zustandes des Kontextes und Entscheidung über die für die jeweilige Situation günstigste Transition zur Veränderung des Zustandes oder Verhaltens eines anzupassenden Objektes.
- Durchführung oder Aktivierung der ausgewählten Transition.

Der Prozess der Kontextadaption lässt sich in drei Teilschritte zerlegen.

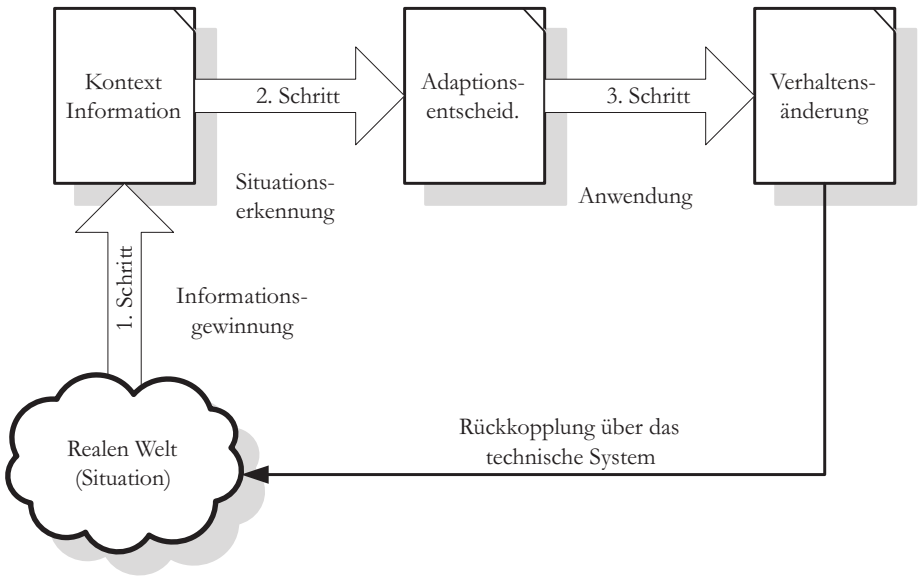


Abb. 16: Teilprozesse der Kontextadaption

Der Gesamtprozess (Abb. 16), in manchen Arbeiten auch *context awareness* [26] genannt, sammelt im ersten Schritt Informationen über die Umgebung, in der sich das betroffene System gerade befindet, und speichert diese in einem Kontext. Der zweite Schritt des Prozesses entscheidet dann, ob der gegenwärtige Kontext eine Situation charakterisiert, die eine Änderung des Systemverhaltens bedingt und welche konkreten Änderungen zu deren Umsetzung notwendig sind. Im letzten Schritt werden die Änderungen dann am System durchgeführt. Indirekt über das technische System als Teil der realen Welt können diese Änderungen auch wiederum die reale Umgebung beeinflussen.

Die Erkennung von Dunkelheit in der Umgebung kann im System einer Lichtsteuerung als Adaption die Aktivierung einer Schaltfunktion auslösen. Die indirekte Folge davon ist die Aktivierung einer Lampe und damit eine Änderung der Helligkeit in der Realität.

Aus den Ubiquitätsbedingungen folgt weiterhin, dass dabei eine möglichst weitgehende Entkoppelung der drei Teilprozesse voneinander gegeben sein muss (für eine detailliertere Anforderungsanalyse siehe [59]).

Der Vorteil dieser Entkoppelung ist, dass die Quelle der Informationen bei Notwendigkeit auch nur partiell angegeben werden kann oder zum Zeitpunkt des Entwurfes auch gänzlich unbekannt sein darf. Die Kommunikation erfolgt zu diesem Zweck indirekt über einen Zwischenspeicher (Abb. 17).

Technisch unterscheidet sich Kontext durch seine lose gekoppelte Kommunikation.

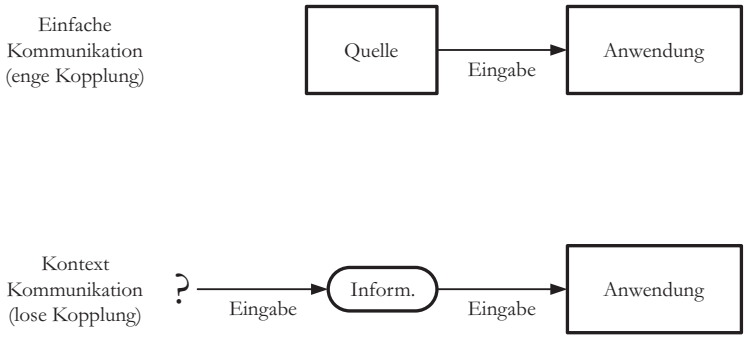


Abb. 17: Entkoppelung von Kontextinformationen

So können die Informationen einerseits zur Laufzeit entdeckt, andererseits zwischen ihrer Erzeugung und Verwendung manipuliert werden. Über diesen Mechanismus ist dann eine nachträgliche indirekte Beeinflussung des Systemverhaltens möglich.

So wurden in der Fallstudie beispielsweise aus systemexternen Quellen gewonnene Umgebungsinformationen vor ihrer Weiterverarbeitung in einem eigenen Datenmodell zwischengespeichert. Auf diese Weise können aus dem Kontext Situationen auch dann zuverlässig erkannt werden, wenn die externen Informationsquellen vorübergehend nicht erreichbar sind, etwa weil eine drahtlose Verbindung kurzfristig gestört ist.

Eine maximale Entkoppelung wird weiterhin erreicht, wenn die Kommunikation von Kontextinformationen über generische Austauschformate (XML etc.) und eine einheitliche Schnittstelle abgewickelt wird. Dies garantiert auch bei der Anbindung neuer Komponenten zur Laufzeit eine gewisse Mindestkompatibilität der Kommunikation. Neue Protokolle können auf diese Weise für die dazwischenliegenden Knoten transparent abgewickelt werden, solange sichergestellt ist, dass sich Sender und Empfänger untereinander verstehen.

Diese Art der technischen Entkopplung und Abstraktion geht über die von dienstbasierten Systemen hinaus. Dort wird aufgrund einer partiellen Be-

schreibung erst zur Laufzeit eine verfügbare Implementierung an eine Komponente der Anwendung gebunden. Im Fall der Kontextkommunikation muss diese Bindung (zwischen zwei Kommunikationspartnern) überhaupt nicht stattfinden. Dadurch wird auch eine Kommunikation zwischen zwei Komponenten möglich, die niemals zur gleichen Zeit existieren. So kann beispielsweise eine Ausgabe initiiert und so lange gespeichert werden, bis ein geeignetes Ausgabemedium zur Verfügung steht. Die eigentliche Quelle muss dann zu diesem Zeitpunkt bereits nicht mehr existieren.

Die einzelnen Schritte einer Kontextadaption können aber über die Entkopplungen an den Teilprozessgrenzen hinaus auch noch weiter in (entkoppelte) Einzelelemente unterteilt werden. Die gesamte Kontextadaption lässt sich so mithilfe von vier Grundelementtypen darstellen:

Vier Grundelemente der Kontextadaption.

- *Sensoren* beschreiben die Quelle einer Kontextinformation und damit den Übergang von der Realität zum Modell.
- *Interpreter* beschreiben Berechnungen auf Kontextinformationen, deren Ergebnis wieder eine Kontextinformation ist.
- *Aktuatoren* beschreiben eine Senke für Kontextinformationen innerhalb des Anwendungssystems und damit den Übergang von Modell zu Realität.
- *Kontextelemente* repräsentieren einzelne oder eine Menge von inhaltlich zusammengehörenden Informationen über die Umgebungssituation, die für das Anwendungssystem relevant sind. Alle Prozesse der Kontextadaption sind über Kontextelemente entkoppelt. Auf diese Weise ist sichergestellt, dass die Ubiquitätsbedingungen (Nutzbarkeit und Verfügbarkeit in jeder Situation) auch bei kurzfristigem Ausfall externer Quellen oder Senken erfüllt bleiben.

Gegenüber vergleichbaren Modellkonzeptionen [26] wurden bei dem vorliegenden Modellierungsansatz lediglich die Aktuatoren hinzugefügt. Dadurch entsteht ein in sich abgeschlossenes Modell ohne direkte externe Abhängigkeiten, wodurch die Definition der für die Kalibrierung notwendigen Reprogrammierbarkeit (der Änderungen am Modell) erheblich erleichtert wird. Weiterhin wurden gegenüber [26] anstelle eines einzigen Kontextservers für die Entkopplung mehrere Kontextelemente eingeführt. Diese Sichtweise erlaubt eine explizite Betrachtung zusammengehörender Umgebungsinformationen als Entitäten (z.B. Kontext eines Raumes, einer Person etc.) konsistent zu Sensoren, Interpretern und Aktuatoren. Kontext bekommt auf diese Weise eine eigene Identität. Dies erleichtert wiederum die Reprogrammierung, aber auch die gemeinsame Darstellung von Daten und Verarbeitungsschritten

in der für die Kalibrierung verwendeten Beschreibungssprache (siehe auch Anforderung nach Multiaspekt-Darstellung in 4.3.1).

5.1.3 Zusammenfassung und Diskussion

Kalibrierung ist der Vorgang der Verwendung einer vom Rest des Anwendungssystems unabhängigen eigenen Benutzerschnittstelle durch den Benutzer zum Zwecke der manuellen Adaption der Kontextadaption. Kontextadaption ist die Menge FP-kritischer Hilfsfunktionen einer Anwendung, welche die Ubiquität der Anwendungsfunktionen herstellen. Da Kontextadaption in der Lage ist, Benutzerschnittstellen zu emulieren, kann die Kalibrierung auch als ein besonderer Spezialfall der Kontextadaption verstanden und modelliert werden.

Die Kontextadaption selbst kann zudem als ein von den zu adaptierenden Funktionen unabhängiger Prozess modelliert werden, der in die Teilschritte Kontextgewinnung, Adaptionsentscheidung und Änderungsdurchführung zerfällt. Jeder Teilprozess wird dabei in Elemente (Sensoren, Kontextelemente, Interpreter und Aktuatoren) modularisiert, welche die technischen Details ihrer Realisierung kapseln (Softwarekomponenten). Alle Elemente und Teilprozesse sind so weit wie möglich voneinander entkoppelt. Dies stellt die Ubiquitätsbedingungen sicher, insbesondere auch im Fall der Kalibrierung (Reprogrammierung der Adaption zur Laufzeit).

Mithilfe der vier Grundelemente lassen sich die unterschiedlichsten Arten und Ausprägungen der Kontextadaption modellieren. Das einfachste Beispiel ist das *Contextual Sensing* [46]. Dem Benutzer wird dabei ein für ihn relevanter Teil der im Kontext einer Anwendung gespeicherten Informationen angezeigt, beispielsweise seine momentane Position auf einer Karte. Eine solche Form der Kontextadaption lässt sich über einen Sensor für die Ortsinformation ausdrücken, der die Daten zunächst in einem Kontextelement ablegt. Von dort kann die Komponente der Kartendarstellung als Aktuator auf die Ortsinformation zugreifen.

Ein Beispiel, dessen Modellierung einen Interpreter benötigt, sind dagegen *Context-Triggered Actions* [30]. Dabei werden bestimmte Funktionen der Anwendung automatisch ausgeführt, sobald ein bestimmter Kontextzustand auftritt, das heißt eine bestimmte Situation erkannt wurde. In einem solchen Fall werden wieder zunächst die Umgebungsinformationen in entsprechenden Kontextelementen zwischengespeichert. Diese werden von einem Interpreter beobachtet. Stellt sich ein bestimmter Zustand ein, der als das Eintreffen einer Situation interpretiert werden kann, wird das jeweilige mit dieser Situation verknüpfte Kommando in ein weiteres Kontextelement geschrie-

ben. Dort wird es von einem Aktuator ausgelesen, der das Kommando ausführen kann.

Etwas schwieriger ist die Modellierung von *Automatic Contextual Reconfiguration* [30]. Damit wird das Hinzufügen oder Entfernen von Softwarekomponenten und Kommunikationsverbindungen zwischen diesen bezeichnet, um beispielsweise Anwendungsprogramme an in bestimmten Räumen fest installierte Aus- oder Eingabegeräte zu binden, solange der Benutzer sich in diesem Raum aufhält. Für die Modellierung einer solchen Kontextadaption ist zunächst ein Sensor notwendig, der die neuen Softwarekomponenten erkennen kann. Interpreter entscheiden dann, welche Teile des Anwendungssystems sich mit den neuen Komponenten verbinden sollen und speichert diese Information zunächst wiederum als Kontext, der die adaptierte neue Struktur der Anwendung beschreibt. Ein geeigneter Aktuator setzt diese Beschreibung dann in entsprechende Nachrichten an die jeweiligen Komponenten um, sich miteinander zu verbinden.

Mithilfe der vier Grundelemente lassen sich aber auch Spezialfälle der Kontextadaption beschreiben, wie beispielsweise die *Contextual Augmentation* [46] oder das Markieren von Informationen durch Kontextbedingungen [26]. Diese Fälle findet man in Systemen, die vornehmlich ortsbasierte Informationen in ihrem eigenen Kontext (oder dem des Benutzers) ablegen. Ähnlich einem Terminkalender oder Touristenführer [21][27], werden diese Nachrichten angezeigt, sobald man einen bestimmten Ort betritt. Die Modellierung solcher Kontextadaptionen ist zunächst eine Context Triggered Action (siehe oben). Der Situationsinterpreter vergleicht den Ortskontext allerdings mit einem anderen Kontextelement, das von der Anwendung selbst verändert werden kann. Dazu muss lediglich ein weiterer Sensor definiert werden, der einen Teil des inneren Zustands der Anwendung (an welchem Ort soll eine Nachricht angezeigt werden) in den Kontext exportiert. Dies ist allerdings kein Widerspruch, da Kontext nicht zwingenderweise nur aus Umgebungsinformationen bestehen darf. Kontext war vielmehr als Modell einer Situation definiert worden und die Situation eines Gegenstandes umfasst auch dessen eigenen Zustand (siehe auch [26]).

5.2 Basismodell Strom verarbeitender Funktionen

Für die formale Definition eines kalibrierbaren Modells für Kontextadaption wurde auf ein allgemeines formales Systemmodell zurückgegriffen, das Systeme als eine Menge von Komponenten und Kanälen modelliert. Es wurde dagegen darauf verzichtet, spezialisierte Kalküle (Ambient [101], Π [100],

etc.) zu verwenden, oder sogar einen eigenen Basisformalismus einzusetzen. Solche Maßnahmen erleichtern in der Regel nur die Beschreibung eines bestimmten Aspektes (Mobilität, Ausführungsorte usw.) oder den Nachweis einzelner Eigenschaften. Andere Aspekte hingegen lassen sich dafür oft schwieriger ausdrücken. Für eine Realisierung müssten solche spezialisierten Spezifikationen zudem zuerst wieder umständlich in eine allgemeine Spezifikation (Programmiersprache o.ä.) transformiert werden. Der Vorteil spezialisierter Kalküle für die Realisierbarkeit hält sich daher in Grenzen. Die beispielsweise im Π-Kalkül ausdrückbare Mobilität ist zwar eine neue explizite Sichtweise, aber keine neue Fähigkeit, die mit bestehenden Sprachen und Formalismen nicht zu realisieren wäre. Das Gleiche gilt natürlich auch für die Kontextadaption.

Da der Fokus dieser Arbeit nicht in den formalen Grundlagen, sondern im Entwurf und der Realisierung (Engineering) ubiquitärer Systeme (darüber hinaus auf der Basis eines konkreten Framework) liegt, werden sowohl die Kontextadaption als auch die Kalibrierung als Elemente höherer Ordnung eines allgemeinen formalen Basismodells (Focus [62][63]) modelliert. Für den Zweck einer klaren Definition einer Semantik des im Framework realisierten Modells kalibrierbarer Kontextadaption (K-Modell) ist das völlig ausreichend. Dies hat darüber hinaus auch den Vorteil der leichteren Zuordnung einer auf Komponenten basierenden formalen Semantik zu einer auf technischen Komponenten basierenden Realisierung des Framework.

In den folgenden Unterabschnitten werden die im nächsten Abschnitt für die Formalisierung der K-Modelle verwendeten Grundlagen und Spezifikationstechniken des Basismodells kurz zusammengefasst.

5.2.1 Grundlagen

Im Basismodell wird ein System als *Komponentennetzwerk* modelliert. Ein solches Netzwerk besteht aus einer Menge von *Komponenten* **C**, die über eine Menge von unidirektionalen asynchronen *Kanälen* **CH** miteinander kommunizieren. Die Komponenten erhalten Zugriff auf die Kommunikationskanäle über *Ports*. Ports werden über einen Kanalbezeichner (x **CH**) und einen Lese-(?) oder Schreiboperator(!) dargestellt. !?**CH** bezeichnet die Menge aller Ports. Einem Kanal kann mithilfe von *type*:**CH** $\mathbf{T_{CH}}$ ein Typ zugeordnet werden, der die Art der übertragbaren Nachrichten definiert. $\mathbf{T_{CH}}$ ist die Menge der Kanaltypen. Über einen Kanaltyp ist festgelegt, welche Art von Nachrichten über einen Kanal fließen können. Für einen Typ t bezeichnet $\|t\|$ die Menge seiner Nachrichten.

Ströme

Das Systemverhalten wird über Relationen auf den Kommunikationshistorien der Ein- und Ausgabekanäle spezifiziert. Eine Kommunikationshistorie wird als Strom von Nachrichten ausgedrückt. Ein Strom $s_n = <m_1, m_2, ...>$ ist eine endliche oder unendliche Sequenz von Nachrichten m_i **M**. Dabei bezeichnet $\mathbf{M}^*$ die Menge der endlichen, **M** die Menge der unendlichen Nachrichtensequenzen. **M** $\overset{def}{}$ $\mathbf{M}^*$ **M** ist die Menge aller Nachrichtenströme über **M**. **M** $\overset{def}{}$ $(\mathbf{M}^*)$ ist die Menge der gezeiteten Nachrichtenströme, dargestellt als unendliche Sequenz von endlichen Sequenzen über **M**.

Kanalbelegung

Zu einer Menge CH getypter Kanäle kann die Kanalbelegung als Zuordnung eines Kanals zu einem Strom von Nachrichten betrachtet werden:

CH $\overset{def}{}$ $\{x$ **CH** (M^*) : c **CH**:$x(c)$ $(\|type(c)\|^*)$ $\}$

Eine Kanalbelegung x **CH** ordnet also jedem Kanal c **CH** einen gezeiteten Strom von Nachrichten aus $type(c)$ zu.

Komponenten

Die Menge aller möglichen Komponenten über einer Menge von Kanälen K **CH** ist definiert durch

$C(K)$ $\overset{def}{}$ $\{(I,O,F)$ $\mathbb{P}(K)$ $\mathbb{P}(K)$ $(\mathbb{P}(K)$ $\mathbb{P}(\mathbb{P}(K))):F$ I $\mathbb{P}(O)\}$

Das Verhalten einer Komponente $c=(I,O,F)$ wird damit als Relation zwischen Eingabe- und Ausgabeströmen modelliert. Dabei ist I die Menge der getypten Eingabe-, O die Menge der getypten Ausgabekanäle. Für ein x I beschreibt F(x) die Menge aller Ausgabehistorien, die eine Komponente mit dem Verhalten F produzieren kann.

Ein Kompositionsoperator für Komponenten ist wie folgt definiert:

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

so dass

I $(I_1$ $I_2) \setminus (O_1$ $O_2)$ und
O $(O_1$ $O_2) \setminus (I_1$ $I_2)$ und

F I $\mathbb{P}(O)$

wobei

$$\begin{array}{cccccccccc} x & I{:}F(x) & \{y' & O{:} & y & I_1 & I_2 & O_1 & O_2{:} \\ y|_O & y' & y|_I & x & y|_{O_1} & F_1(y|_{I_1}) & & y|_{O_2} & F_2(y|_{I_2})\} \end{array}$$

Dabei ist $x|_C$ die Einschränkung der Belegung x auf die Kanäle in C.

5.3 Formalisierung von Adaption als rückgekoppelte Filter

Die Formalisierung der K-Modell Modellierung selbst besteht im Wesentlichen aus zwei Bestandteilen.

- Erstens einer Definition der Adaption und ihrer Kalibrierung durch Reprogrammierbarkeit.
- Zweitens einer Definition der in der Modellbildung 5.1 skizzierten technischen Elemente Sensor, Kontext, Interpreter und Aktuator zur Strukturierung einer Adaption in Form eines K-Modells.

5.3.1 Adaption

Im Basismodell werden Systeme oder Teilsysteme als ein Netz von über Kanäle miteinander kommunizierender Komponenten dargestellt. Deren Verhalten ist durch eine Relation zwischen den Ein- und Ausgabeströmen auf ihren Kanälen, also dem beobachtbaren Verhalten genauer festgelegt. Adaption lässt sich in diesem Modell zunächst rein anschaulich als eine Änderung des Komponentennetzwerks verstehen, welches das Softwaresystem modelliert. Komponenten oder Kanäle können hinzugefügt oder entfernt und das Verhalten des Gesamtsystems dadurch verändert werden.

Die Formalisierung und auch die Realisierung dieser Art von Systemdynamik ist allerdings kompliziert. Der Grund dafür ist, dass prinzipiell jede Art von Modell als statisch anzusehen ist. Ein Modell entsteht durch Abstraktion der Realität (siehe Definition 5 in 3.1.1) wobei die Abstraktionen Annahmen sind, dass sich etwas (Struktur, Relevanz, etc.) nicht verändert (statisch ist). Ein Modell besteht also immer aus einer statischen Einschränkung oder Vereinfa-

chung der Wirklichkeit. Dynamik kann auf diese Weise aber nicht wirklich dargestellt, sondern lediglich durch statische Annahmen angenähert werden.

Die Auswirkungen dieser sehr fundamentalen Aussage sind uns ja bereits in Form des Frame-Problems begegnet, lassen sich aber auch an der ganz alltäglichen Erfahrung demonstrieren, dass die Zukunft nicht exakt vorhergesehen, sondern höchstens statistisch angenähert werden kann. Selbst diese Annäherung ist nur auf der Basis der Annahme möglich, dass kleine Ereignisse die Wirklichkeit nicht grundlegend verändern und ähnliche Ursachen ähnliche Wirkungen erzielen (starke Kausalität). Diese Annahme gilt aber nur für bestimmte Teile der Wirklichkeit. Andere Teile verhalten sich dagegen chaotisch oder können zeitweise in einen unvorhersehbaren chaotischen Zustand wechseln (Wetter, Doppelpendel, etc.).

Für eine Formalisierung von Dynamik bedeutet dies, dass sie sozusagen durch ein statisches Modell emuliert werden muss. Emuliert bedeutet in diesem Zusammenhang, dass sich das Modell lediglich nach außen hin beobachtbar so verhält, als ob es dynamisch wäre.

Das macht man sich auch ganz leicht anschaulich klar. Ein laufendes Softwaresystem kann sich nur so verhalten, wie es spezifiziert wurde. Soll sich sein Verhalten ändern, kann das nur in einer Weise geschehen, die vorher spezifiziert wurde. Da die Spezifikation aber das Verhalten festlegt, bedeutet das, dass sich das Verhalten in Wirklichkeit nicht ändert. Für einen Beobachter kann dennoch der perfekte Eindruck entstehen, als hätte sich das (beobachtbare) Verhalten verändert, weil plötzlich ein Teil des Gesamtverhaltens sichtbar wird, der vorher nicht beobachtet werden konnte. Umgekehrt können vorher sichtbare Verhaltensmuster möglicherweise nicht mehr länger beobachtet werden.

Modellierung von Dynamik nur durch Wechsel zwischen Ausschnitten eines beobachtbaren Verhaltens möglich.

Das letzte Beispiel gibt auch gleich eine Antwort auf die Frage, wie man mit statischen Modellen Dynamik und insbesondere Adaption formal spezifiziert, wenn das Verhalten eines Systems eigentlich nicht veränderbar ist. Die Emulation der Dynamik erzielt man einfach durch die Konstruktion eines Modells, welches Dynamik durch einen Wechsel zwischen der Sichtbarkeit verschiedener Ausschnitte eines internen Gesamtverhaltens darstellt.

Ein adaptives System muss folglich formal als Superposition aller möglichen Strukturen, Funktionen und Verhaltensweisen betrachtet werden. Die eigentliche Adaption erfolgt dann durch einen Filter, der nach außen hin lediglich die Beobachtung eines bestimmten Ausschnitts dieser Superposition erlaubt (Abb. 18).

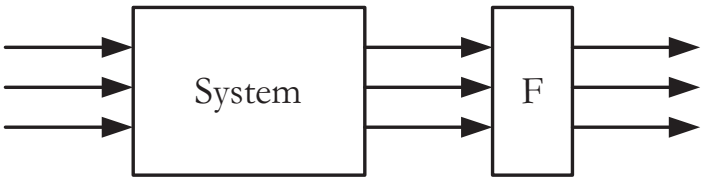


Abb. 18: Adaption als Ausgabefilter

Abb. 19 zeigt, dass die Beschränkung auf einen Ausgabefilter ausreichend ist, um die anschauliche Vorstellung von Adaption also das Hinzufügen und Entfernen von Komponenten und Kanälen zu emulieren.

Formal lässt sich eine für die Adaption geeignete Filterkomponente aus einer Kombination von Einzelfiltern F_{A0} für jeden Ausgabekanal des Systems spezifizieren.

F_{A0}

in	i : MSG
out	o: MSG
p	{0,1} :o⊒filter$_1$(*p*,i)

Wobei filter$_j$ {0,1,..,n} MSG MSG eine Hilfsfunktion ist mit

$$a=j \Longrightarrow \text{filter}_j\,(a\&p,b\&i)=b\&\text{filter}_j\,(p,i)$$

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

Jeder dieser Einzelfilter kontrolliert einen Ausgabekanal gemäß einer nicht näher spezifizierten Filterlogik, die in der Formalisierung als Orakel *p* dargestellt wurde. Ist der n-te Wert dieses Orakels 0, wird damit die n-te Nachricht des Eingabekanals i der Filterkomponente herausgefiltert. Ist er dagegen 1, wird die n-te Nachricht auf dem Ausgabekanal o ausgegeben.

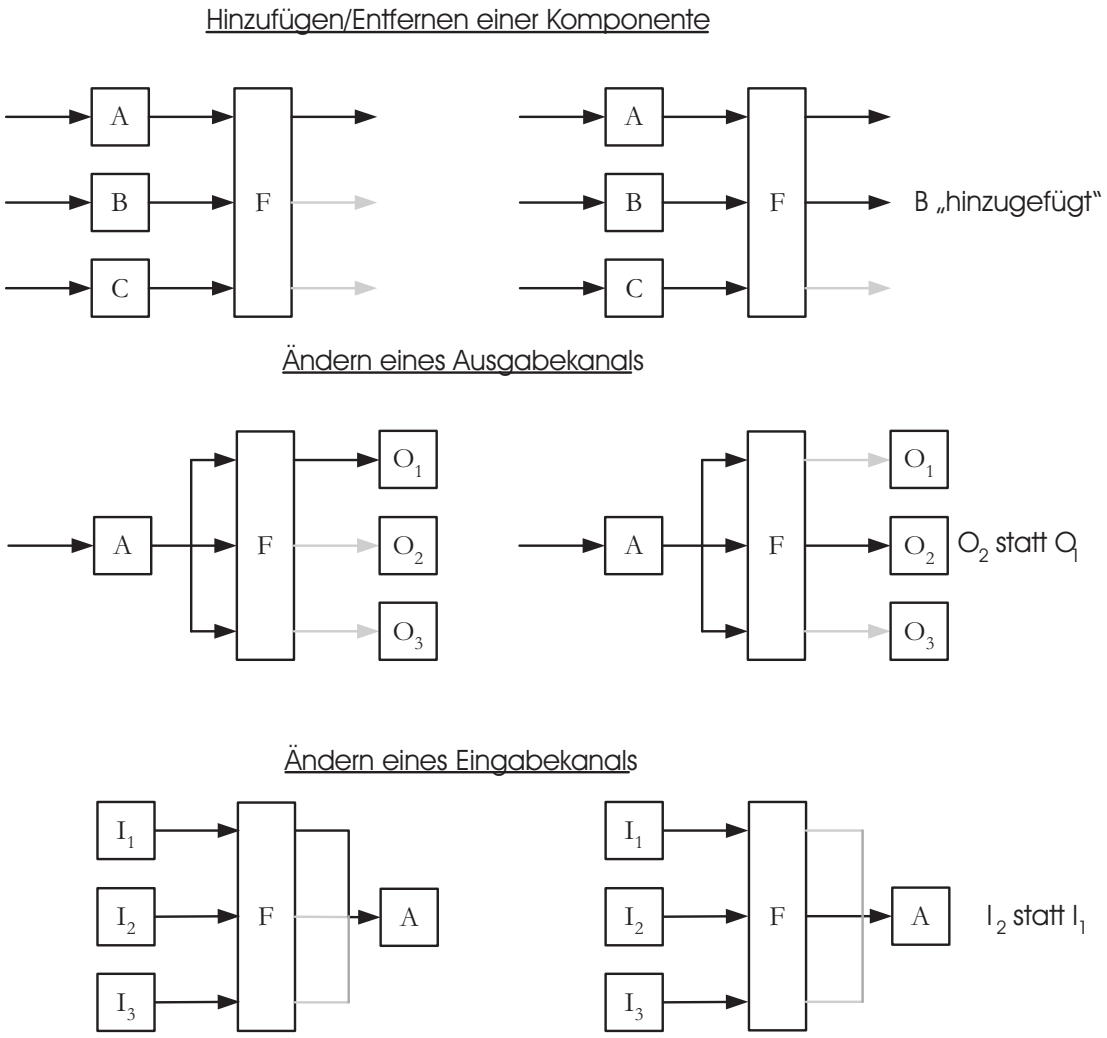


Abb. 19: Emulation von Adaptionsdynamik durch Filterung

Die Hilfsfunktion $filter_j$ verwendet dafür einen Strom *i* der zu filternden Nachrichten von einem Eingabekanal sowie einen Strom von Steuerinformationen *p*. Entspricht die n-te Nachricht im Steuerstrom dem Filtercode j, wird die entsprechende Nachricht des Eingabestroms ausgegeben, andernfalls verworfen.

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

Für eine Komponente C mit *n* Ausgabekanälen ist deren Adaption damit gegeben durch $F_{A_C} \quad F_{A0_1} \quad ... \quad F_{A0_n}$.

5.3.2 Kontextadaption

Kontextadaption, also die Adaption in Abhängigkeit von bestimmten Umgebungsinformationen, kann als Erweiterung der allgemeinen Adaption formalisiert werden (Abb. 20).

Bei der Kontextadaption steuert ein Kontext den Ausgabefilter.

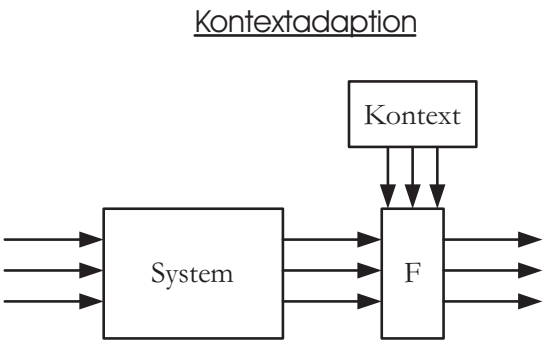


Abb. 20: Kontextadaption als gesteuerter Filter

Dafür werden dem Filter eine Menge von Steuerkanälen hinzugefügt. Für jeden zu filternden Kanal gibt es einen Steuerkanal, mit dem die Filterung eines Ausgabekanals kontrolliert wird. Die Steuerkanäle werden aus Komponenten gespeist, die den Kontext des zu adaptierenden Systems bilden.

Formal kann die Kontextadaption spezifiziert werden, in dem die in F_{A0} enthaltenen Orakel für die Filterlogik auf einen Steuerkanal ausgelagert werden.

F_{A1}

in i : MSG , s : {0,1}	
out o: MSG	
#o=#1 $(s\|_{\#i})$	; Die Anzahl der Nachrichten auf o (#o) entspricht der Anzahl der "1" Nachrichten aus dem Steuerkanalstrom beschränkt auf die Länge der Eingabenachrichten $(s\|_{\#i})$
$o \sqsupseteq filter_1(s\|_{\#i}, i)$	

Die Kontextadaption lässt sich wie die allgemeine Adaption aus einer Kombination der Einzelfilter bilden.

5.3.3 Kalibrierung

In der Modellbildung wurde argumentiert, dass sich die Kalibrierung auch als eine Adaption der Kontextadaption betrachten lässt. Das macht man sich auch leicht mithilfe der bisherigen Formalisierungen klar:

- Das Adaptionsverhalten wird im Fall der Kontextadaption durch die Informationen auf den Steuerkanälen bestimmt (Abb. 20). Diese stammen aus dem Kontext.
- Kalibrierung bedeutet nun, das an den Steuerkanälen sichtbare Verhalten des Kontextes zu verändern. Der Kontext wird also adaptiert (Abb. 21).

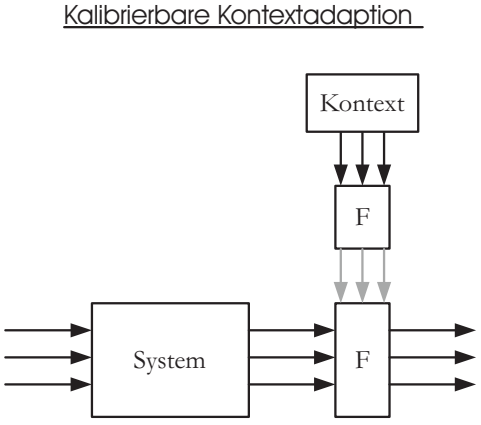


Abb. 21: Kalibrierung als Adaption des Filters

Die für die Kalibrierung notwendige zweite Adaption wird zweckmäßig wiederum als Kontextadaption modelliert (Abb. 23).

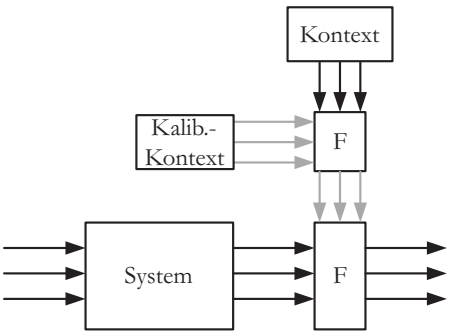


Abb. 23: Kalibrierung als wiederholte Kontextadaption

Damit ist zwar die erste Kontextadaption veränderbar, die für die Kalibrierung verwendete zweite Kontextadaption ist aber nach wie vor statisch und unterliegt damit möglicherweise dem Frame-Problem. Dies ist der Fall, sofern die Ausgaben an den Steuerkanälen aus den Eingabekanälen des Kalibrierungskontextes interpretiert werden und nicht direkt von einem Benutzer stammen. In 3.3 war bereits diskutiert worden, dass es in gewissen Fällen aber sinnvoll sein kann, die Kalibrierung (teilweise) zu automatisieren, beispielsweise um mehrbenutzertaugliche oder lernfähige Anwendungen zu realisieren. Wegen des Frame-Problems muss eine solche automatisierte Kalibrierung wie jede andere Adaption wiederum kalibrierbar sein. Bei der Formalisierung müsste der in Abb. 23 abgebildete Schritt daher je nach Anwendung mehrfach ausgeführt werden. Die Formalisierung ist dann aber nicht mehr generisch anwendbar. Stattdessen bietet es sich an, eine Möglichkeit zu konstruieren, die Kalibrierung rekursiv auf sich selbst anzuwenden. Dazu müssen die beiden Kontextadaptionen miteinander kombiniert werden. Im ersten Schritt werden dazu die beiden Kontexte kombiniert (Abb. 22).

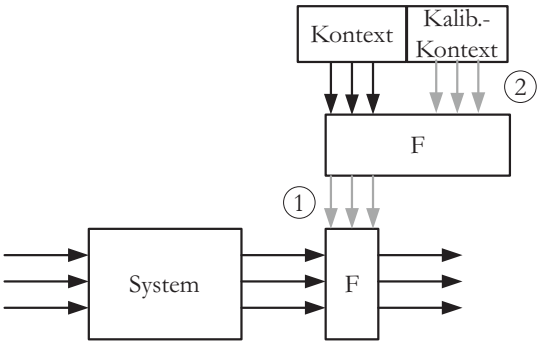


Abb. 22: Kombination von Kontext und Kalibrierungskontext für Selbstadaption

Sowohl die alternativen Steuerinformationen für die Adaption des Hauptsystems als auch die Steuerinformationen zu deren Auswahl stammen nun aus einem gemeinsamen Kontext. Damit kann der Kontext sich selbst adaptieren, genau genommen aber lediglich einen Teil von sich selbst. Um sich vollständig selbst adaptieren zu können, müsste auch bei den Steuerinformationen, welche die Kalibrierung kontrollieren (Abb. 22-2) zwischen mehreren Alternativen ausgewählt werden können. Dies entspräche dem Wechsel zwischen mehreren Eingabekanälen wie in Abb. 19 veranschaulicht. Mit der bisher definierten Filterfunktion für Kontextadaption F_{A1} ist das aber (im Gegensatz zur allgemeinen Adaption durch F_{A0}) nicht mehr möglich, da für jeden alternativen Kanal ein weiterer Steuerkanal hinzugefügt werden müsste.

Um eine rekursive Adaption oder auch ganz generell den in Abb. 19 veranschaulichten Wechsel zwischen mehreren Eingabekanälen mithilfe einer kontextabhängigen Filterkomponente zu formalisieren, muss F_{A1} daher so modifiziert werden, dass eine ganze Gruppe alternativer Kanäle durch einen einzigen Steuerkanal adaptiert werden kann.

$F_{A2(n)}$

in $i_1,...i_n$: MSG , s : {0,1,...,n}
out $o_1,...,o_n$: MSG

k {1,..n}:#o_k=#k $(s|_{\#i_k})$

k {1,..n}: $o_k \sqsupseteq filter_k(s|_{\#i_k}, i_k)$

Um mit dem Filter F_{A2} den in Abb. 19 dargestellten Wechsel zwischen Eingabekanälen zu erreichen, muss der Filterkomponente genau genommen noch eine Zusammenfassung (Merge) der gefilterten Ausgabekanäle zu einem einzigen Kanal nachgeschalten werden. Die Formalisierung einer solchen Mergefunktion unter Beibehaltung der richtigen Reihenfolge und unter Abstraktion einer Zeittaktung ist jedoch problematisch, wenn nicht auf den Steuerkanal von F_{A2} zurückgegriffen werden kann. Es ist daher sinnvoll, die Mergefunktion in den Filter F_{A2} zu integrieren:

$F_{A3(n)}$

in $i_1,...i_n$: MSG , s : {0,1,...,n}
out o: MSG

k {1,..n}: $filter_k(s|_{\#i_k}, i_k) \sqsupseteq filter_k(\{1,..n\}$ s,0

Dazu werden einfach alle Ausgaben auf einen einzigen Ausgabekanal gemultiplext. Für die spätere technische Verwendung dieser Adaptionsfunktion ist dabei wichtig, dass für den Ausgabestrom nicht nur zwischen verschiedenen Eingabekanälen ausgewählt, sondern die Ausgabe auch vollständig unterbunden werden kann (Nachricht "0" auf dem Steuerkanal).

Für n=2, $i_1'=filter_1(s,i_1)$, $i_2'=filter_2(s,i_2)$ und $s'=\{1,2\}$ s ergibt sich beispielsweise folgende Arbeitsweise von $F_{A3(n)}$.

```
i1   : abcdefghijklmnopqrst...
i1'  : behknqt...
i2   : ABCDEFGHIJKLM...
i2'  : CFIL...
s    : 012012012012012012...
s'   : 12121212121212...
o    : bCeFhIkL...
```

Der bisherige Filter F_{A1} für Kontextadaption kann auch in allen anderen Fällen durch die neue Komponente F_{A3} ersetzt werden. Allerdings lässt sich die Filterkomponente nicht mehr wie bisher auf Basis einzelner Kanäle in Einzelfilter zerlegen, sondern nur noch auf Basis der jeweils von einem Steuerkanal kontrollierten Kanalgruppe. Die Einteilung in Kanalgruppen erfolgt zweckmässigerweise so, dass ein Steuerkanal alle Kanäle kontrolliert, die zu alternativen Komponenten mit gleicher syntaktischer Ausgabeschnittstelle gehören.

Mit F_{A3} ist nun zwar eine theoretische Rückkopplungsmöglichkeit des Ausgabekanals auf den Steuerkanal und damit eine Rekursionsmöglichkeit gegeben, wie sie für die Kalibrierung der Kontextadaption notwendig ist. Bei einer Rückkopplung auf den Steuerkanal ist F_{A3} aber ein geschlossenes System. Dies bedeutet, es kann sich entweder selbst kalibrieren oder eine andere Kontextadaption, nicht aber beides gleichzeitig. Zu diesem Zweck müssten mindestens zwei F_{A3} Komponenten so kombiniert werden, dass sie von einem einzigen Steuerkanal kontrolliert werden können. Dieser müsste sowohl die für die Filterkontrolle der ersten F_{A3} Kanalgruppe (die Selbstkalibrierung) notwendigen Informationen enthalten als auch die Steuerinformationen für die Filterung der zweiten (oder n weiterer) F_{A3} Kanalgruppen, die nicht rückgekoppelt werden, sondern als Nutzinformation zur Verfügung stehen.

$F_{A4(k,n1,..nk)}$

in $i_{11},..i_{1n_1},..,i_{kn_k}$: MSG ,
s: $\{(s_1,..,s_k) | s_1 \quad \{0,1,...,n_1\},..,s_k \quad \{0,1,...,n_k\}\}$
out $o_1,..o_k$: MSG

m $\{1,..k\}$: $F_{A3}(i_{m1},..i_{mn_m}, elemstrm_m(s), o_m)$

Wobei

$elemstrm_j\{(s_1,..,s_k) | s_1 \quad \{0,1,...,n_1\},..,s_k \quad \{0,1,...,n_k\}\} \qquad \{0,1,...,n_k\}$

eine Hilfsfunktion ist mit

$0 \quad j \quad k \quad elemstrm_j((s_1,...,s_k)\&x) = s_j \& elemstrm_j(x)$,

welche die Steuerinformationen einer Kanalgruppe $i_{k1},..,i_{kn_k}$ aus dem Gesamtsteuerkanal s extrahiert und an einen F_{A3} Filter für diese Kanalgruppe weiterleitet, der den dazu gehörenden Ausgabekanal o_k kontrolliert.

Für k=2 (zwei Kanalgruppen), n_1=2, n_2=2 und s'=$elemstrm_1${s} sowie o_1 für die Selbstkalibrierung rückgekoppelt auf s ergibt sich beispielsweise folgende Arbeitsweise von F_{A4}.

```
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    : abcdefghijklmnopqrst...
i22    : ABCDEFGHIJKLM...
o1     : (1,0)(1,1)(2,2)(2,1)...
o2     : bCd...
```

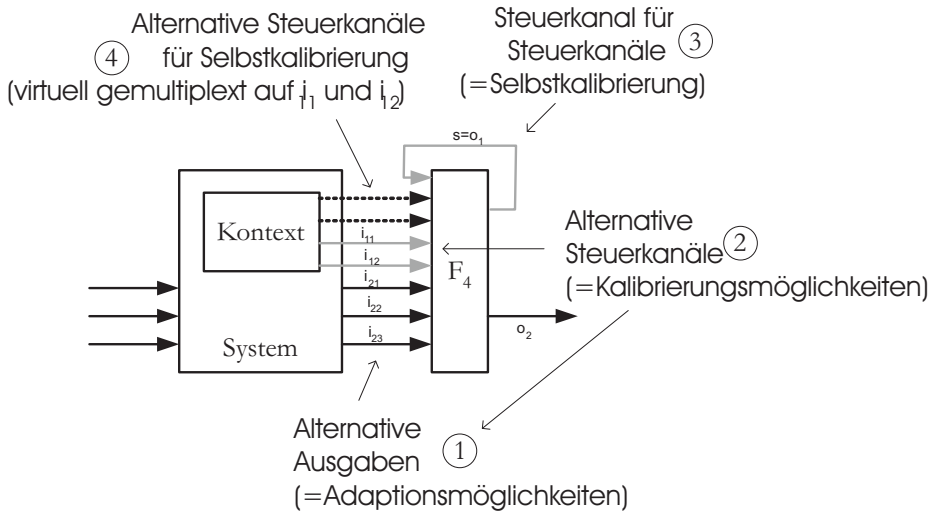


Abb. 24: Kalibrierung durch rekursive Adaption

Mithilfe dieser rekursiven Adaption F_{A4} kann für die Steuerung der Kalibrierung auf einen rekursiven Steuerkanal (Abb. 24-3) zurückgegriffen werden. Dieser rekursive Kanal legt fest, welcher der alternativen Kalibrierungssteuerungen (Abb. 24-4) für die Adaption der Steuerkanäle (Abb. 24-2) verwendet wird, welche die Adaption des Anwendungssystems (Abb. 24-1) kontrollieren. Der jeweils aktive Kanal der Kalibrierungssteuerung (aus der Kanalgruppe 1 wird gleichzeitig auf den rekursiven Steuerkanal der Selbstkalibrierung rückgekoppelt. Auf diese Weise kann eine Kalibrierungssteuerung im Kontext sozusagen die Kalibrierungskontrolle an eine andere Alternative übergeben.

5.3.4 Technische Verfeinerungen

Über die Filterfunktion F_{A4} mit ihrer Rückkopplungsmöglichkeit auf sich selbst, ist die kalibrierbare Kontextadaption formal eindeutig definiert. Im Hinblick auf die im nächsten Kapitel gezeigte technische Realisierung der Formalisierung als Framework und der darauf abgestützten Beschreibungstechnik und Methodik müssen lediglich noch die folgenden Punkte diskutiert werden:

- Wie passen eine möglicherweise unendliche Anzahl von spezifizierten alternativen Komponenten und Komponentenstrukturen auf eine endliche technische Realisierung?
- Wie können die in der Modellbildung 5.1 genannten drei Teilprozesse der Kontextadaption auf die Formalisierung abgebildet werden?

- Nach welchen Regeln lassen sich die einzelnen Schritte wie in der Modellbildung überlegt, in die Elemente Sensor, Interpreter, Aktuator und Kontext unterteilen und wie werden diese spezifiziert?

Spezifikation vs. Implementierung

Es gibt einen Unterschied zwischen Spezifikation und Implementierung.

Hinsichtlich einer technischen Realisierung ist zunächst zu klären, wie die Spezifikation einer möglicherweise unendlichen Anzahl von alternativen Komponenten und Komponentenstrukturen überhaupt technisch zu verwirklichen ist. Auf den ersten Blick erscheint diese Festlegung überdies widersprüchlich, zumindest bezogen auf die Kalibrierung. Wenn nämlich alle möglichen Arten von Adaptionsverhalten bereits im Voraus spezifiziert sein müssen, kann zur Laufzeit durch den Benutzer kein wirklich neues Adaptionsverhalten hinzugefügt werden, sondern es wird lediglich ein vorbereitetes Verhalten ausgewählt.

Die Auflösung dieses nur bei oberflächlicher Betrachtung erscheinenden Widerspruchs ist allerdings sehr einfach. Die Spezifikation eines Verhaltens ist nicht automatisch gleichbedeutend mit der Existenz einer Implementierung zum Entwurfszeitpunkt des Systems. Genau genommen ist die Spezifikation vieler alternativer Komponenten und Strukturen ein Modell, das möglichst wenig statische Annahmen enthält. Je weniger Abstraktionen desto komplexer wird das Modell. Ein auf diese Weise spezifiziertes Verhalten enthält dann fast die Menge aller möglichen, von außen beobachtbare Teilverhalten zuzüglich einer Möglichkeit, zwischen diesen zu wechseln.

Adaption erlaubt partielle Implementierung.

Die explizite Modellierung dieses Wechselmechanismus und die Festlegung der Ausschnitte des Gesamtverhaltens, zwischen denen gewechselt wird, ist aber gerade das, was in dieser Arbeit als Adaption bezeichnet wird. Die Verknüpfung der Steuerung der Adaption mit einem Kontext, erlaubt dem Systementwickler überhaupt erst, Teile der Gesamtspezifikation des Systems unimplementiert zu lassen. Über die Kontextadaption kann er nämlich sicherstellen, dass zu einem bestimmten Zeitpunkt nach außen hin nur ein klar definiertes Teilverhalten exponiert wird, das genau den Möglichkeiten der momentanen existierenden Implementierung entspricht. Ohne die explizite Modellierung eines Wechselverhaltens zwischen Verhaltensausschnitten gibt es dagegen keine realistische Möglichkeit, streckenweise inaktive Partitionen einer Systemspezifikation zu erkennen, bei denen temporär auf eine Implementierung verzichtet werden kann.

Eine solche Festlegung klingt aus technischer Sicht abstrakt, ist aber einfach zu veranschaulichen. Ein vollständig reprogrammierbares und damit maximal adaptives System ist so spezifiziert, dass es nach außen hin einen beobachtbaren Ausschnitt seines Verhaltens erzeugen kann, der jeweils genau der Eingabe (Programmierung) an einem bestimmten Kanal entspricht.

Aus dieser Betrachtungsweise wird auch klar, dass die Adaption nur im Zusammenhang mit einer Möglichkeit Sinn macht, ein spezifiziertes System zur Laufzeit automatisch zu implementieren oder zumindest automatisch mit einer bereits existierenden Implementierung zu verbinden. In der Softwaretechnik stehen dazu bereits verschiedene Konzepte dynamischer Implementierung bereit. Am bekanntesten ist das dynamische Binden von Methoden und Funktionen im Rahmen von objektorientierten Vererbungskonzepten (z.B. *virtual* und *override* Methoden) oder das dynamische Nachladen von Bibliotheken bei Bedarf (DLL etc.). Die flexibelste Technik dynamischer Implementierung steht aber mit dem Konzept der Dienste (engl. *Services*) zur Verfügung, das sogar einen Wechsel zwischen Komponentenimplementierungen zur Laufzeit erlaubt.

Dienste und partielle Rekonfiguration

Unter dem Begriff Rekonfiguration versteht man in der Softwaretechnik die An- oder Umordnung der Kommunikationsstruktur einer vorgegebenen Menge von Komponentenimplementierungen, um eine bestimmte Aufgabe (Funktion, Anforderung) zu erfüllen. Für einen Teilausschnitt der Menge, der lediglich anhand von ähnlichem Verhalten und unter Abstraktion der Komponentenidentitäten gebildet wird (logische Architektur) kann dies auch als "Austausch" einzelner Komponentenimplementierungen betrachtet werden.

Der Begriff der Rekonfiguration in Zusammenhang mit Softwarekomponenten stammt eigentlich aus der Telekommunikation. Er geht zurück auf die Notwendigkeit, die Kommunikationsanlagen amerikanischer Militärfahrzeuge an die Gegebenheiten und Bestimmungen unterschiedlicher Einsatzorte anzupassen. Dies geschah üblicherweise durch den Austausch (Rekonfiguration) kompletter Hardwaremodule. Um diese teure und aufwändige Prozedur zu vereinfachen, wurden verschiedene Projekte in Auftrag gegeben, den Vorgang in Software zu realisieren (*Software Defined Radio*, SDR).

Die zivile Telekommunikationsindustrie erkannte schnell die kommerzielle Bedeutung (Kosten sparende Realisierung von Mulit-Mode Terminals) und nahm die Anregung in verschiedenen Forschungsprojekten auf (z.B. [90][91][92]).

Eine geeignete softwaretechnische Methode zur Umsetzung von Software Rekonfiguration auf Basis von Komponenten sind *Dienste* (eng. *Services*). Hinter Diensten verbirgt sich die Idee, die Funktion von Komponenten von ihrer Realisierung zu abstrahieren. Anwendungen sollen damit lediglich über ihre funktionalen Abhängigkeiten (logische Architektur) spezifiziert und implementiert werden. Zur Laufzeit wird dann aus einem existierenden Angebot an implementierten und instanzierten Komponenten (mit wiederum unterschiedlichen Abhängigkeiten) eine geeignete Konfiguration ausgewählt und an die logische Architektur gebunden, so dass die Funktionalität der Anwendung realisieren werden kann (*Design@Runtime* [89]).

Für die formale Spezifikation von Diensten können die Modellierungsmöglichkeiten des Basismodells zum Beispiel um das Element einer nach innen strukturlosen Komponente (Dienst) erweitert [12] werden. Damit lässt sich das Teilverhalten einer Komponente bezüglich einer Signatur beschreiben und explizit an ein oder mehrere Komponenten binden, die dieses Teilverhalten realisieren.

Ein Dienst wird in [12] mithilfe eines Teilverhaltens F_i' einer Komponente dargestellt, das in Relation zu einer Teilmenge $I_i' \quad I, O_i' \quad O$ ihrer Signatur I,O definiert wird. Das Teilverhalten darf sich dabei gegenseitig nicht beeinflussen.

Definition 12: Komponente *erfüllt* Dienst

Eine Komponente c=(I,O,F) wird als einen *Dienst erfüllend* bezeichnet, wenn es einen Dienst d=(I_D,O_D,F_D) mit $I_D \quad I, O_D \quad O$ gibt, so dass für eine Funktion $F':I' \quad \mathbb{P}(O')$ mit $I' \quad I \setminus I_D$ und $O' \quad O \setminus O_D$ gilt:

Sei $F_D : I_D \quad \mathbb{P}(O_D)$, dann gilt

$F':I' \quad \mathbb{P}(O') \quad F \quad F_D \quad F' \; I_D \quad I' \quad O_D \quad O'$.

Insbesondere bedeutet dies, dass die von einer Komponente erfüllten Dienste immer disjunkte Kanäle besitzen[12].

Diese Art der Definition eines Dienstes ist allerdings noch sehr wenig konkret, denn sie besagt lediglich, dass ein Dienst genau genommen eine Komponente ist, die sich durch andere ersetzen lässt. Nichts anderes aber verbirgt sich hinter der Dienstidee auf der technischen Ebene. Die reale Nutzung eines Dienstes ist eine Implementierung, die sich zur Laufzeit durch eine andere ersetzen lässt.

Allerding wird durch den Dienstbegriff aus [12] nicht klar, wie dieser Ersetzungsvorgang genau definiert ist. Für die vorliegende Arbeit wurde der Dienstbegriff daher um die folgenden Definitionen erweitert:

Definition 13: *Agent*

Ein Agent ist die Instanz einer Komponente c=(I,O,F) und besitzt einen eindeutigen Identifikator *a ID*. Die Menge aller Agenten über einer Menge von Kanälen *K* **CH** ist definiert durch

$$A(K) \overset{def}{} \{(a,I,O,F) \quad ID \quad \mathbb{P}(K) \quad \mathbb{P}(K) \quad (\mathbb{P}(K) \quad \mathbb{P}(\mathbb{P}(K))):F \quad I \quad \mathbb{P}(O)\}$$

a A(CH) mit a=(id,I_D,O_D,F_D) definiert einen einzelnen Agenten.

Diese Definition hilft dabei, zwischen Komponenten und Implementierungen zu unterscheiden. Technisch gesehen können über ein Blackbox-Verhalten definierte Komponenten auf verschiedene Weise implementiert werden. Eine einzige Implementierung kann zusätzlich mehrfach instanziert werden. Formal ist aber schwierig zu definieren, ob mehrfache Instanzierungen eigentlich unterschiedliche Implementierungen darstellen. Für die Definition des Austauschens einer (technischen) Komponente ist dieser Unterschied aber nicht von Belang. Es muss lediglich eine Möglichkeit bestehen, auch zwischen zwei Instanzen einer identischen (formalen) Komponente zu wechseln.

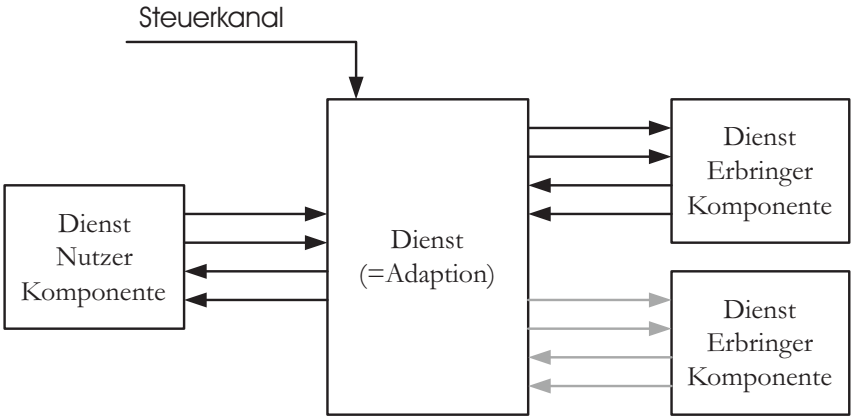


Abb. 25: Der Dienstbegriff als Adaption

An dieser Stelle ergibt sich der Zusammenhang zu der in 5.3.1 formal definierten Adaption. Das Austauschen einer technischen Komponente kann auch als Wechsel zwischen alternativen Ein- und Ausgabekanälen und damit als Adaption betrachtet werden (siehe Abb. 25).

In Wirklichkeit handelt es sich um zwei getrennte Adaptionen. Einmal die Adaption der Ausgabekanäle der Dienst-Nutzerkomponente bezüglich alternativer Ausgabeziele (die Dienst erbringenden Komponenten). Zum zweiten einer Adaption der Rückantworten der alternativen Diensterbringer.

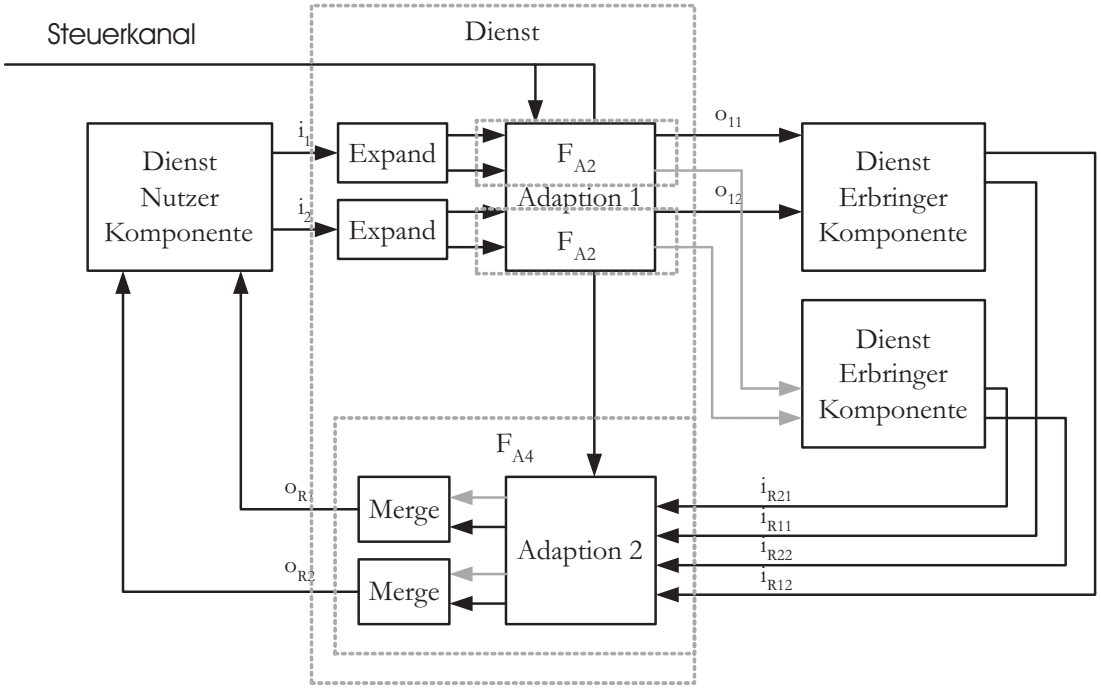


Abb. 26: Betrachtung eines Dienstes als zwei Adaptionen

Strukturiert man Abb. 25 in diese zwei Adaptionen, erhält man den in Abb. 26 gezeigten Zusammenhang. Man erkennt sofort, dass sich die Adaption der Dienstaufrufe in mehrere Filterkomponenten des Typs F_{A2} (Filterung einer Gruppe Kanäle durch einen gemeinsamen Steuerkanal) zerlegen lässt, sofern die Kanäle des Dienstaufrufes vorher geeignet vervielfacht worden sind und die Steuerkanäle der Kanalgruppen aus dem gemeinsamen Steuerkanal extrahiert werden.

Die Adaption der Rückkanäle entspricht dagegen bereits fast vollständig der Filterkomponente F_{A4} (Adaption von Kanalgruppen mit integriertem Merge). Es muss lediglich auch hier die Steuerinformation geeignet aufbereitet werden.

Aus diesen Überlegungen ergibt sich die Formalisierung einer Dienstkomponente wie folgt:

D(n,m,k)

in $i_1,..i_n$: MSG ,	; Aufrufkanäle 1..n
$i_{R11},..i_{R1m},..,i_{Rkm}$: MSG ,	; Rückkanäle 1..m von 1..k Komponenten
s : {0,1,...,k}	; Bindung an 1..k Diensterbringer oder 0
out $o_{11},..o_{1n},..,o_{km}$: MSG	; gefilterte Ausgabekanäle 1..n zu 1..m
	; Diensterbringerkomponenten
$o_{R1},..o_{Rm}$: MSG	; gefilterte und gemergte Rückkanäle 1..m

j {1,..n}: $F_{A2(k)}(\overbrace{i_j,...i_j}^{k},s,o_{j1},o_{jk})$	; Aufrufkanäle werden dupliz. u. ; für eines von 1..k Ziele aktiviert
h {1,..m}: $F_{A3(k)}(i_{R1h},..i_{Rkh},s,o_{Rh})$	; Rückkanäle von einem v. 1..k ; Diensterbringern werden ; freigeschaltet

Ein Dienst ist damit eine Art von Proxy-Komponente, welche die Dienstaufrufe transparent für die Dienstnutzungskomponente an eine von k alternative Diensterbringungskomponenten weiterleitet und eventuelle Rückantworten so an den Aufrufer weitergibt, als würde dieser lediglich mit einer einzigen Komponente kommunizieren.

Definition 14: *Dienst*

Die Menge aller Dienste D ist definiert als die Menge aller Komponenten $c=(I_D \quad \{s\},O_D,D(|I_D|,|O_D|,k)),k\in\{0,..,|a|\}$, welche den Proxy Zugriff (D) auf die Menge aller Agenten a=(id,I,O,F) aller Komponenten erlaubt, welche ein Teilverhalten F_D erfüllen, also c=(I,O,F) und es gilt $c=c_D \quad c_1$ mit $c_D=(I_D,O_D,F_D)$. Weiterhin definiert d D mit $d=(I_D,O_D,F_D,k)$ einen einzelnen Dienst, um sich im Folgenden auf ein einzelnes Element beziehen zu können.

Definition 15: *Rekonfiguration*

Das Rekonfigurationsverhalten eines Dienstes wird durch den Nachrichtenstrom am Steuerkanal *s* erzeugt. Die Rekonfiguration kann daher als eine weitere Komponente dargestellt werden.

Bei der Reprogrammierung eines Softwaresystems über den Austausch von Komponentenimplementierungen gibt es bestimmte Invarianten, die sich in Form der logischen Architektur (der Dienste) des betrachteten Teilausschnitts ausdrücken. Eine völlige Umordnung (totale Rekonfiguration) der Kommunikationsstruktur auf einer gegebenen Menge von Komponenten-

implementierungen ist daher in der Regel nicht möglich. Durch die Dienste ist ein bestimmtes beobachtbares Verhalten an den Grenzen des betrachteten Teilausschnittes (also eines Systems) gegeben. Dieses invariante Verhalten kann auch als die erfüllte Funktion, Aufgabe oder Anforderung betrachtet werden, die von ihrer Realisierung abstrahiert wurde.

Zurückkommend auf die ubiquitären Anwendungen mit den beiden Ubiquitätsbedingungen kann mit Diensten und der dadurch erzeugten partiellen Rekonfigurierbarkeit die Verfügbarkeit bestimmter Funktionen sichergestellt werden, die von externen Ressourcen aber nicht von konkreten Implementierungen abhängen. Dies gilt, solange sich in einer gegebenen Situation eine Kombination der verfügbaren Komponenten finden lässt, welche die definierte Funktion erfüllt.

Zum Beispiel ist eine ubiquitäre Anwendung damit nicht von einem konkreten Bildschirm oder Drucker abhängig, sondern kann diese als Dienste realisieren, die zur Laufzeit an eine lokale Instanz einer Drucker- oder Bildschirmkomponente gebunden wird.

Gibt es mehr als eine solche Kombination für die Realisierung, spricht man von Freiheitsgraden bei der Bindung eines Dienstes oder Rekonfigurationsmöglichkeiten. Will man Dienste für die Kalibrierung, also die Reprogrammierung der Kontextadaption einsetzen, bedeutet dies, dass man zwischen diesen Freiheitsgraden wählen kann. Obwohl man damit das insgesamt beobachtbare Gesamtverhalten der Adaption nicht verändern kann, kann man sehr wohl deutliche Unterschiede im Verhalten bezogen auf eine einzige Situation erzielen.

Stehen beispielsweise für die Adaption einer Klimaregelung zwei Temperaturfühler zur Verfügung, von denen sich einer im Schatten, der andere in der Sonne befindet, kann das im Ergebnis der Adaption einen deutlichen Unterschied erzielen.

Aktivator und Totalrekonfiguration

Für die Kalibrierung von Kontextadaption ist die partielle Rekonfiguration über Dienste allerdings nicht in allen Fällen ausreichend. Die aus Diensten aufgebaute logische Architektur einer Kontextadaption kann nämlich ebenfalls bestimmte Interpretationen (Annahmen) der Wirklichkeit enthalten, etwa bestimmte Abhängigkeiten und Abfolgen in der Erkennung einer Situation.

Ein schönes Beispiel dafür war in der Fallstudie 2.4 beschrieben. Dort wurde zunächst nur die Implementierung der Lichtadaption verändert, so dass sie zusätzlich das Signal eines manuellen Schalters berücksichtigte. An der prinzipiellen Abfolge (wenn Signal, dann schalten) änderte sich jedoch nichts. Dadurch konnten Situationen in denen beide Signale (das automatische aus Helligkeit und Bewegung sowie der Schalter) fast gleichzeitig betätigt wurden nicht richtig interpretiert werden.

Dies entspricht auch den genaueren Untersuchungen[78] des Frame-Problems, die in solchen Fällen die komplette Änderung der im Modell enthaltenen Interpretationen oder Annahmen, also seine möglichst vollständige Reprogrammierung verlangen. Wie eingangs dieses Abschnitts diskutiert, kann dies aufgrund der prinzipiell statischen Natur von Modellen aber lediglich durch Auswahl aus einem möglichst allgemeinen internen Verhalten geschehen. Auf Rekonfiguration und Dienste übertragen bedeutet dies, dass man die völlige Reprogrammierbarkeit durch eine Superkomponente erzielt, deren Aufgabe darin besteht, ein von außen beschriebenes Teilverhalten durch einen geeigneten Ausschnitt ihres internen Gesamtverhaltens zu reproduzieren. Dies erreicht man mit den folgenden Definitionen:

Definition 16: *Rekonfigurierbarer Dienst*

Ein Dienst ist rekonfigurierbar, wenn die den Dienst darstellende Zugriffskomponente über einen den Steuerkanal deaktiviert und aktiviert werden kann.

Zu diesem Zweck wurde das Verhalten der formalen Definition eines Dienstes D bereits so festgelegt, dass eine Bindung an einen leeren Agenten (Nullkomponente) erfolgen kann. Ist der Dienst deaktiviert, bedeutet dies, dass keine Bindung an einen Agenten existiert. Die empfangenen Nachrichten (ausgenommen am Steuerkanal) werden verworfen, bis am Steuerkanal eine Bindung!=0 empfangen wird.

Die Steuerung der Aktivierung rekonfigurierbarer Dienste entspricht letztendlich der in 5.3.3 definierten Kalibrierung einer Adaption, einschließlich der Möglichkeit zur Selbstkalibrierung. Im Gegensatz zu der in 5.3.3 mit F_{A4} gegebenen Gesamtdefinition kalibrierbarer Adaption soll nun aber auf die mit Diensten vorgegebenen Teiladaptionen zurückgegriffen werden. Es muss also aus F_{A4} die Logik der Aufspaltung der Steuerinformationen samt ihrer Bereitstellung herausgelöst werden. Die Möglichkeit zur Selbstadaption wie-

derum bedeutet, dass diese Kalibrierungslogik in Form eines Dienstes formuliert werden muss (Abb.27).

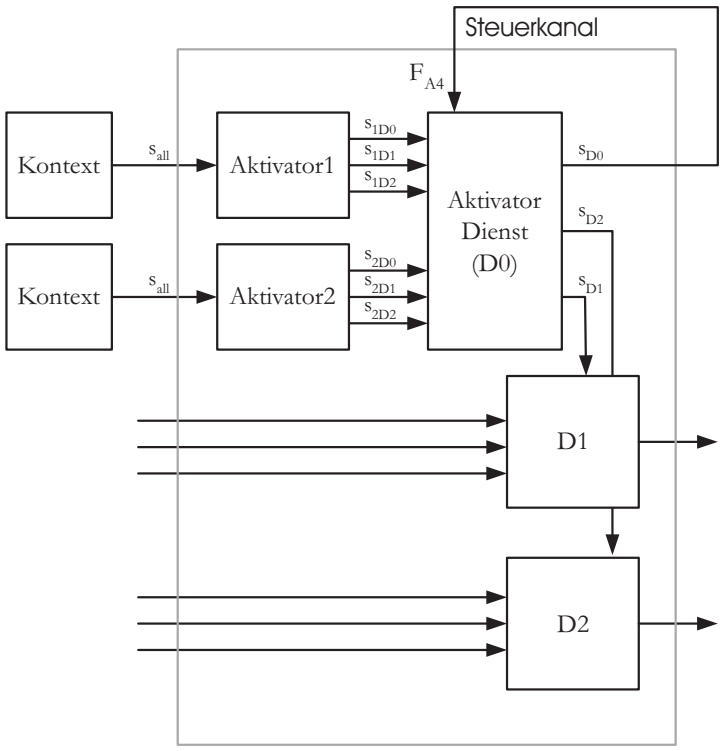


Abb. 27: Darstellung kalibrierbarer Adaption als Dienste

Die Spezifikation einer Aktivatorkomponente ergibt sich aus der von ihm kontrollierten Dienstmenge:

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

in: s_{all}: $\{(s_1,..,s_k) | s_1 \quad \{0,1,...,n_1\},..,s_k \quad \{0,1,...,n_k\}\}$
out s_{D0}: $\{0,1,...,n_1\}$,...,s_{Dk}: $\{0,1,...,n_k\}$

$m \quad \{0,..k\}: s_{Dm} = \text{elemstrm}(s_{all})$

Definition 17: *Aktivator* über einer Dienstmenge

Der Aktivator erzeugt aus einem gemeinsamen Steuerkanal s_{all} die Steuerinformationen für alle Dienste der Dienstmenge, einschließlich für einen Aktivatordienst, dessen Diensterbringer er selbst ist.

Unter der Voraussetzung, dass die möglichen Abhängigkeiten von Komponenten einer gegebenen Menge von Agenten (Komponentenimplementierungen) wiederum als Dienste spezifiziert sind, kann durch den Aktivator so-

wohl das Ändern einer logischen Architektur, als auch der Austausch von Komponentenimplementierungen und die Änderung deren Kommunikationsverbindungen untereinander emuliert werden.

Nach der Definition der Rekonfiguration (s.o.) kann das Rekonfigurationsverhalten des Aktivators (der Eingabestrom am Steuerkanal) wiederum als eigene Komponente modelliert werden. Gemäß den Festlegungen in 5.3.3 kann diese Komponente als der Kontext bezeichnet werden.

5.3.5 K-Modell

Die bisherige formale Definition der kalibrierbaren Kontextadaption und auch ihre technische Interpretation auf der Basis von Diensten und deren Kalibrierung durch einen Aktivator hat dennoch zwei Nachteile. Erstens ist die Modellierung einer konkreten Kontextadaption für ein konkretes Anwendungssystem noch sehr komplex, da unter Umständen sehr unterschiedliche Arten von Diensten vorkommen können. Zweitens ist die in der Modellbildung angeregte Strukturierung der Kontextadaption in drei auf dem Kontext operierende Schritte noch nicht erkennbar.

Hier hilft aber eine Rückbesinnung auf die ursprüngliche Zielsetzung. Diese sah eine Zweiteilung des Systems in seine Adaption und in seine Kernfunktionen vor. Die Kalibrierung also die Reprogrammierbarkeit war nur für das Teilsystem der Adaption gefordert worden. Dies wiederum bedeutet, dass der Aktivator nur Dienste kalibrieren können muss, welche Komponenten adaptieren, die direkt die Adaption realisieren. Deren weitere Abhängigkeiten müssen nicht durch den Aktivator kontrolliert werden. Es verbleiben also lediglich Dienste, welche an Komponenten gebunden sind, die der Erzeugung, Zwischenbearbeitung und Verwendung von Kontext dienen, sowie den Kontext selbst realisieren. Dies entspricht genau den vier in der Modellbildung genannten Rollen *Sensor*, *Interpreter*, *Aktuator* und *Kontextelement*. So gehört der Aktivator beispielsweise der Aktuatorrolle an.

Zudem wurde in der Modellbildung gefordert, dass Komponenten in diesen Rollen in ihrer Kommunikation voneinander zu entkoppeln sind. Dies bedeutet, dass sie nicht direkt miteinander, sondern ausschließlich über den Kontext kommunizieren. Dies wiederum bedeutet, dass die den Rollen zugeordneten Dienste eine einheitliche Schnittstelle besitzen können. Somit kann die kalibrierbare Kontextadaption unabhängig von der jeweiligen Anwendung aus lediglich vier festen Diensttypen konstruiert werden.

Definition 18: *K-Modell*

Ein K-Modell ist das Modell einer kalibrierbaren Kontextadaption, das aus einem Aktivator über einer Dienstmenge von Sensoren, Interpretern, Aktuatoren und Kontextelementen besteht.

Kontextelemente sind dabei Dienste, welche das folgende Teilverhalten spezifizieren:

F_{ctx}

in i:(CTX T) ,CTX MSG,T ℕ
; Kontextinformation und Zeit-
; Stempel schreiben
r:T ,T ℕ ; Kontextinformation zu Zeit-
; Stempel lesen

out o:CTX

k, j {0,..#i};k j;$(c_1,t_1),(c_2,t_2)$ *CTX* *T*:
(c_1,t_1) pos(i,k) (c_2,t_2) pos(i,j) t_1 t_2

o⊒gen(i,r)

Wobei gen (CTX T) T CTX eine Hilfsfunktion ist mit

gen(i,a&r) = choose(expand(i,0,⊥),a)&gen(i,r),

welche aus dem Eingabestrom `i` die Kontextnachrichten gemäß einem bestimmten Zeitstempel auswählt und zu einem Ausgabestrom anordnet, der einem Eingabestrom `r` von Zeitstempeln entspricht.

Die Hilfsfunktion choose (CTX T) T CTX

$t_1=t \Longrightarrow$ choose((c_1,t_1)&e,t)=c_1

$t_1>t \Longrightarrow$ choose((c_1,t_1)&e,t)=choose(e,t)

$t_1<t \Longrightarrow$ choose((c_1,t_1)&e,t)=⊥

übernimmt dabei die Auswahl des Kontextwertes zu einem bestimmten Zeitpunkt. ⊥ ist der undefinierte Kontextzustand.

expand (CTX T) T CTX (CTX T) ist eine Hilfsfunktion, mit

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

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

Sie ergänzt einen Strom von Kontextnachrichten mit Zeitstempeln um die Zwischenwerte, indem die Kontextnachrichten fortgeschrieben werden, bis eine andere Kontextnachricht mit einem höheren Zeitstempel die Gültigkeit der letzten Nachricht beendet (z.B. aus (a,1),(b,3) wird (a,1), (a,2), (b,3))

pos CTX T CTX ist eine Hilfsfunktion, mit

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

$p=1 \Longrightarrow pos(a\&i,p)=a,$

welche das Element an einer bestimmten Position des Stroms liefert.

Neben den Kontextelementen spezifizieren Sensoren, Interpreter und Aktuatoren Dienste, die ein Lese- oder Schreibverhalten auf ein Kontextelement beschreiben:

$F_{sen(n)}$

out $o_1,..o_n$:(CTX T) ,CTX MSG, T $\mathbb{N}$

Ein Sensor ist ein beliebiges partielles Komponentenverhalten, das auf den Ausgabekanälen Kontextnachrichten erzeugen kann.

$F_{act(n)}$

in $i_1,..i_n$:CTX **out** $r_1,..r_n$:T,
k {1,..n}:$\#i_k=\#r_k$

Aktuatoren sind beliebige Komponenten, die Kontextnachrichten aus einem Kontextelement abfragen können. Dazu müssen sie in der Lage sein, Zeitstempelnachrichten für die Abfrage zu erzeugen.

$F_{int(n,m)}$

in $i_1,..i_n$:CTX **out** $r_1,..r_n$:T, $o_1,..o_m$:(CTX T) ,CTX MSG, T $\mathbb{N}$
k {1,..n}:$\#i_k=\#r_k$

Interpreter vereinen die Funktionalität eines Sensors und eines Aktuators. Sie dienen dazu, Kontextnachrichten zu interpretieren und in Form anderer Kontextnachrichten wieder in Kontextelementen zu hinterlegen.

KAPITEL 6 Framework

Wie werden K-Modelle technisch realisiert?

Das folgende Kapitel beschreibt die Realisierung eines kalibrierbaren Modells der Kontextadaption (K-Modell), wie es im letzten Kapitel formal spezifiziert worden war. Da Kalibrierung eine möglichst vollständige Reprogrammierbarkeit zur Laufzeit bedeutet, ist jede Realisierung eines K-Modells (funktional betrachtet) für jede beliebige ubiquitäre Anwendung geeignet. Die Realisierung hat damit den Status eines Framework oder einer Middleware. Diese muss lediglich noch mit dem richtigen initialen Adaptionsverhalten initialisiert und mit den Implementierungen der jeweiligen Anwendungsfunktionen (die auch zur Laufzeit aus der Umgebung entdeckt werden können) kombiniert werden, um eine fertige kontextadaptive Anwendung zu erhalten.

Definition 19: *Framework* _____

Ein Framework ist eine wiederverwendbare, "semivollständige" Applikation, die spezialisiert werden kann, um eine maßgeschneiderte Anwendung zu erhalten [87].

Die nächsten Abschnitte dieses Kapitels beschreiben die Realisierung der wichtigsten Bestandteile eines K-Modells. Dazu gehören das Modell selbst, Hilfsfunktionen für seine Reprogrammierung, abstrakte Schnittstellen der Anwendungsfunktionen sowie eine abstrakte Benutzerschnittstelle für die Kalibrierung mit einer Beschreibungstechnik, die dem Benutzer die Reprogrammierung der Kontextadaption ermöglicht.

Das nächste Kapitel beschreibt dann eine Methodik, wie auf der Basis dieses Framework eigene ubiquitäre Anwendungen und deren Kalibrierung entworfen werden können.

6.1 Realisierung des Modells und seiner Reprogrammierung

In 5.3 war dargestellt worden, wie das kontextadaptive Verhalten einer konkreten ubiquitären Anwendung formal als ein Komponentennetzwerk spezifiziert werden kann. Dort wurde auch beschrieben, wie mithilfe von rekonfigurierbaren Diensten und einem Aktivator die Reprogrammierbarkeit eines solchen Netzwerks formal definiert wird. Der Aktivator benötigt dazu lediglich eine Liste von aktiven Diensten und deren gültige Komponentenbindungen. Abb. 28 zeigt das Modell einer solchen Beschreibung, die im Zuge der Kalibrierung vom Benutzer an den Aktivator übermittelt wird.

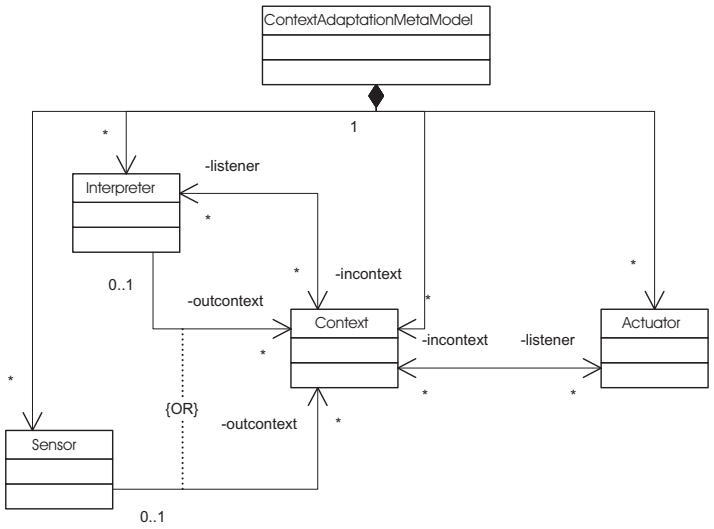


Abb. 28: Metamodell einer Kontextadaption

Der Kontext ist sein eigenes Metamodell.

In 5.1.1 war erklärt worden, warum es sinnvoll und möglich ist, einen solchen Kalibrierungsvorgang wiederum als eine Kontextadaption zu betrachten. Der Benutzer definiert über eine Benutzerschnittstelle (Sensor) eine Kalibrierungsbeschreibung (Abb. 28). Diese wird zunächst in einem Kontextelement gespeichert und dort später vom Aktivator (Aktuator) ausgelesen und ausgeführt. Das betroffene Kontextelement kann daher auch als Realisierung eines Metamodells (Modell für Spezifikationen von Kontextadaption) betrachtet werden.

6.1.1 Realisierung eines Kontextelementes

Ein Kontextelement ist gemäß der Definition in 5.3.5 genau genommen nichts weiter als ein Zwischenspeicher für beliebige Daten. Ein Kontextelement wird daher als ein Container für ein sich selbst beschreibendes Daten-

format realisiert. Im Beispiel des vorliegenden Framework handelt es sich um ein XML-Dokument.

Kontextelemente sind Container für sich selbst beschreibende Datenformate.

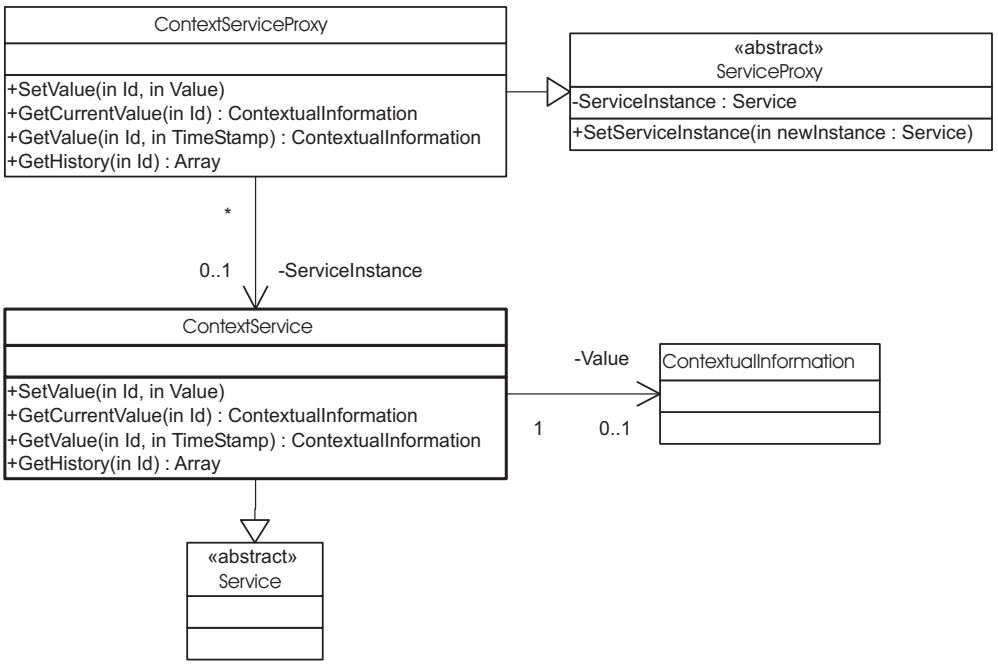


Abb. 29: Realisierung eines Kontextelementes

Um die Reprogrammierbarkeit zu gewährleisten, wurde ein Kontextelement in 5.3.5 ebenfalls als total rekonfigurierbarer Dienst spezifiziert. Das bedeutet, dass eine Komponente, die ein so spezifiziertes Kontextelement realisiert, immer über eine spezielle Proxy-Komponente angesprochen wird (Abb. 29). Dieser Proxy steht mit allen[1] möglichen Implementierungen des von ihm repräsentierten Dienstes in Verbindung und kann daher zur Laufzeit zwischen diesen verschiedenen Realisierungen wechseln (Adaption). Da Implementierungen von Kontextelementen aber im Gegensatz zu Sensoren, Interpretern und Aktuatoren einen Zustand besitzen, darf ein solcher Wechsel zur Laufzeit nur unter bestimmten Bedingungen erfolgen (dazu später mehr).

Bei der Realisierung von Kontextelementen wurde gegenüber der formalen Definition in 5.3.5 lediglich von der Möglichkeit gebrauch gemacht, mehrere gleichartige Kanäle zu einem einzigen Kanal zusammenzuführen (der Identifikator *Id* erlaubt die Separation der gemultiplexten Kanäle). Der Hintergrund ist, dass laut Definition eine Komponente auch mehrere Dienste er-

[1] Aufgrund der dynamischen Eigenschaften objektorientierter Modellierungen und den late-binding Fähigkeiten ihrer Laufzeitumgebungen genügt hier im Gegensatz zur formalen Modellierung eine einzige Referenz im Serviceproxy, um verschiedene Implementierungen anzusprechen.

bringen kann, solange keine Kanäle gemeinsam genutzt werden. Davon möchte man bei der Realisierung der Kontextelemente natürlicherweise Gebrauch machen, da die Realisierung jedes Kontextelementes durch eine einzelne Komponente sehr ineffizient wäre. Stattdessen kann man aber auch die Realisierung mehrerer Kontextelemente zusammenfassen. Für jedes Kontextelement existiert dann zwar nach wie vor ein eigener Zugriffs-Proxy, diese können aber auf eine gemeinsame Serviceinstanz (eine Art von Kontextserver) zugreifen. Da aber die Realisierung einer unbegrenzten Anzahl von separaten Zugriffskanälen nicht möglich ist, greift man auf die Lösung zurück, die Kanäle virtuell durch Multiplexing auf einen einzigen Kanal abzubilden und im Inneren des Kontextservers wieder zu teilen.

6.1.2 Sensoren, Interpreter, Aktuatoren

Sensoren, Interpreter und Aktuatoren werden im Framework als abstrakte Klassen realisiert. Aus Gründen der Reprogrammierbarkeit werden davon abgeleitete Komponenten wiederum als total rekonfigurierbare Dienste und damit über einen Serviceproxy verwendet.

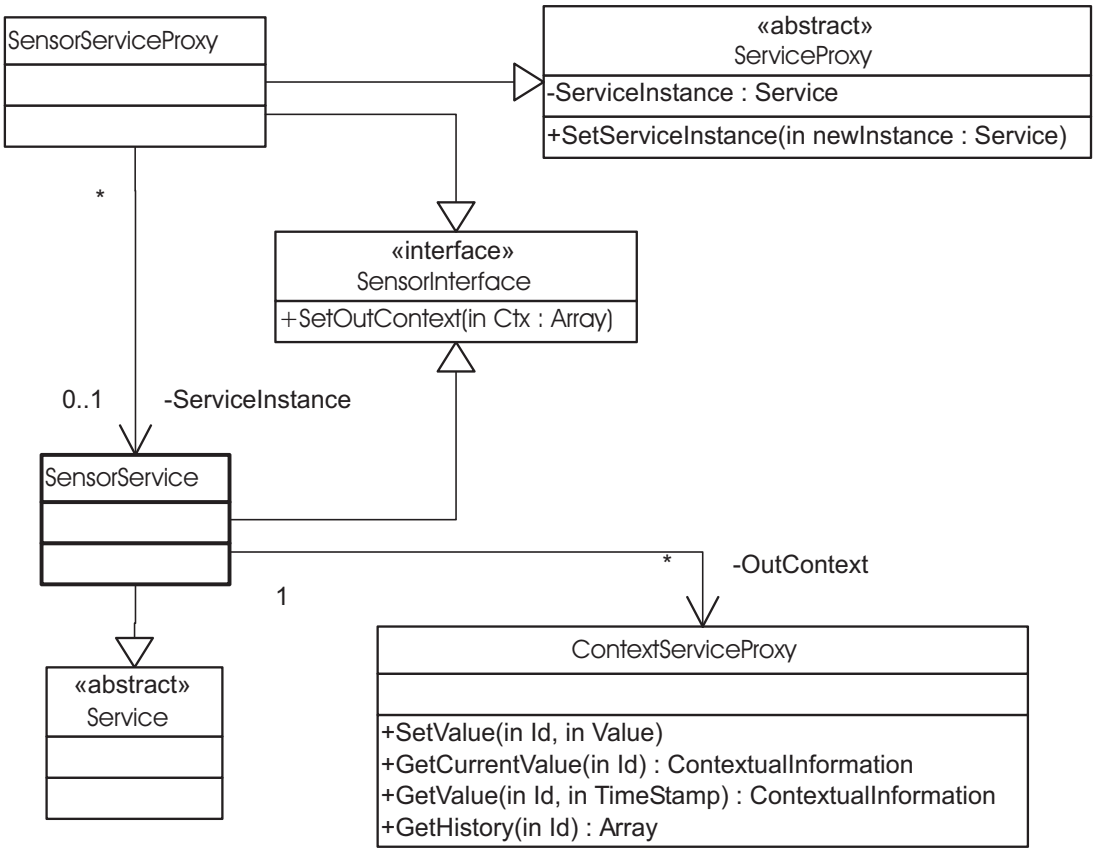


Abb. 30: Realisierung eines Sensorelementes

Vor einer Verwendung müssen die abstrakten Klassen je nach Art der einzubindenden Funktionalität konkretisiert werden. Der Zugriff auf den jeweiligen Dienst (der Serviceproxy) gehört dabei immer zum Teilsystem der Kontextadaption.

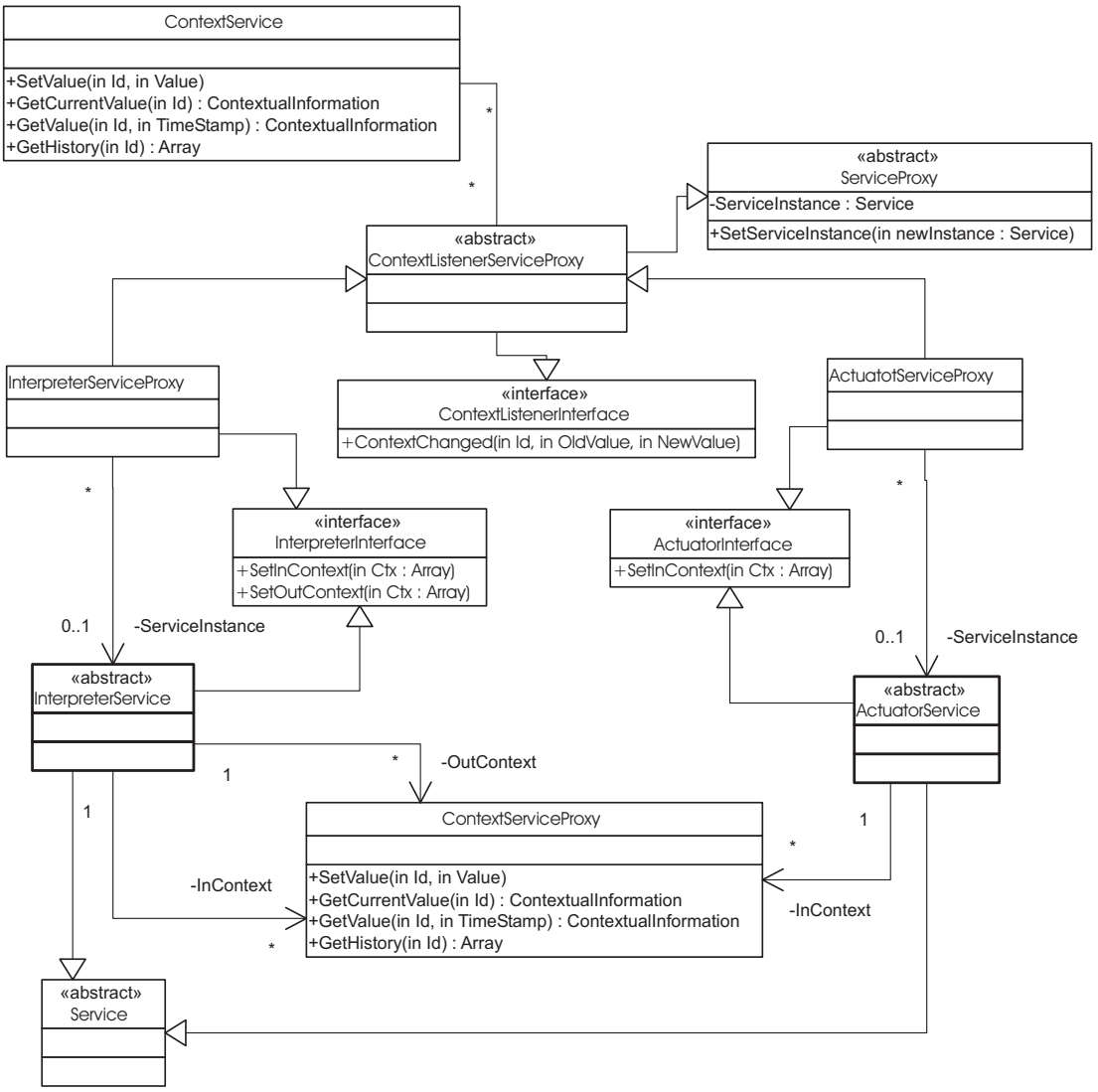


Abb. 31: Realisierung von Aktuator und Interpreterelementen

Die eingebundene Funktionalität kann hingegen entweder extern vorliegen oder Teil der Anwendungsfunktionen des Systemkerns sein. Im letzteren Fall wird der Zugriffs-Proxy im Teilsystem der Kontextadaption also an eine diensterfüllende Komponente aus dem Kernsystem gebunden, im ersteren Fall an eine zur Laufzeit entdeckte Komponente aus der Umgebung.

6.1.3 Aktivator

Für die Realisierung der in 4.1 geforderten Reprogrammierbarkeit durch totale Rekonfiguration fehlt im Framework noch die Realisierung des in 5.3.5 spezifizierten Aktivators. Dieser ist eine Art von Super-Proxy, der mit allen theoretisch möglichen Instanzen von `ServiceProxy` verbunden ist. Seine Aufgabe besteht lediglich darin, diese Service-Proxies (Repräsentanten von Diensten auf der Ebene der Komponenten) gemäß einer über einen Eingabekanal empfangenen Liste zu aktivieren oder zu deaktivieren, beziehungsweise deren Komponentenbindung zu verändern. Auf diese Weise können Komponenten und Kanäle zwischen diesen zur Laufzeit hinzugefügt oder entfernt werden.

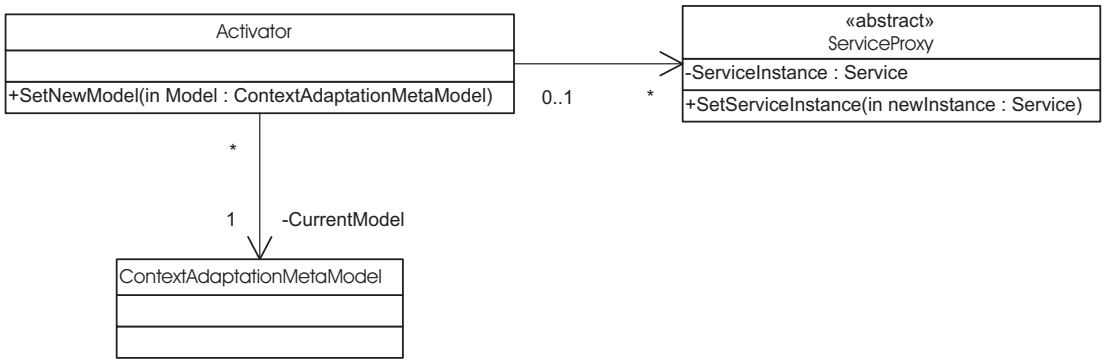


Abb. 32: Mögliche Realisierung des Aktivators

Im Framework akzeptiert der Aktivator eine solche Liste aktiver Dienste und ihrer Komponentenbindungen in Form des in Abb. 28 definierten Metamodells. Dass damit trotz der statischen Spezifikation eine Dynamik modelliert wird, es also möglich ist, zur Laufzeit das Teilsystem der Kontextadaption zu reprogrammieren, macht man sich leicht an folgendem Beispiel klar:

Das in Abb. 28 definierte Metamodell enthält sowohl Identifikatoren für Dienste, als auch Identifikatoren für die gerade jeweils daran gebundenen Komponenten. Ein solcher Identifikator kann nicht nur eine Referenz, sondern auch ausführbaren (oder compilierbaren) Code enthalten. Der Aktivator (compiliert und) instanziert diesen Code und erzeugt eine neue Referenz, bevor er den Dienst (den ServiceProxy) aktiviert, oder die Komponente an den Dienst bindet.

Soll eine Komponente entfernt werden, wird ihre Referenz gelöscht (null) und ihr Identifikator (Code) entfernt. Dadurch können tatsächlich neue Komponenten (und damit auch Funktionen) zur Laufzeit gebunden werden. Dynamische Classloader in vielen modernen objektorientierten Programmiersprachen (Java, C#, etc.) arbeiten im Übrigen nach dem gleichen Prinzip.

6.1.4 Realisierung des K-Modells und seiner Kalibrierung

Die über den Aktivator aggregierte Menge aller Sensor-, Interpreter-, Aktuator- und Kontextelement-Serviceproxy-Instanzen stellt die Realisierung eines konkreten kalibrierbaren Modells der Kontextadaption (K-Modell) dar und bildet damit gleichzeitig das Teilsystem einer konkreten ubiquitären Anwendung, welches deren kontextadaptives Verhalten kapselt.

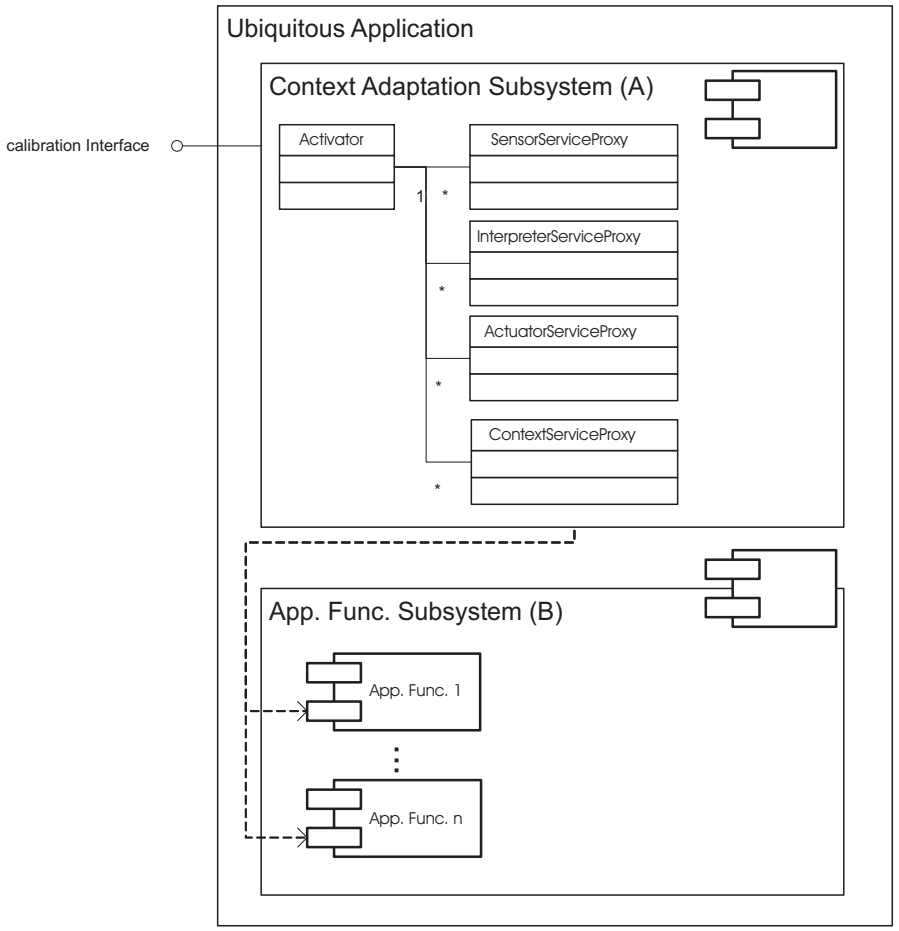


Abb. 33: Entwurfsmuster Kontextadaption - K-Modell und Anwendungskern

In 5.1.1 war aber auch klargemacht worden, dass die Kalibrierung als Kontextadaption in einigen Fällen durchaus wieder Interpretationen der Situation (automatische Funktionen) enthalten kann, beispielsweise für das schnelle Umschalten zwischen den Kalibrierungen verschiedener Benutzer. Solche Interpretationen sind wiederum potenziell FP-kritisch und bedürfen einer Kalibrierung. Technisch lässt sich eine solche rekursive Abhängigkeit am besten dadurch lösen, dass die Realisierung eines K-Modells sich selbst adaptieren kann. Die Kalibrierung selbst findet über die Schnittstelle des Aktivators statt (Abb. 33). Der Kalibrierungsvorgang wird dazu noch einmal als eine eigene Kontextadaption betrachtet. Diesmal allerdings bilden die Sensor-, Interpreter-, Aktuator- und Kontext-Serviceproxies (Teilsystem A in Abb. 33) das zu adaptierende Anwendungssystem.

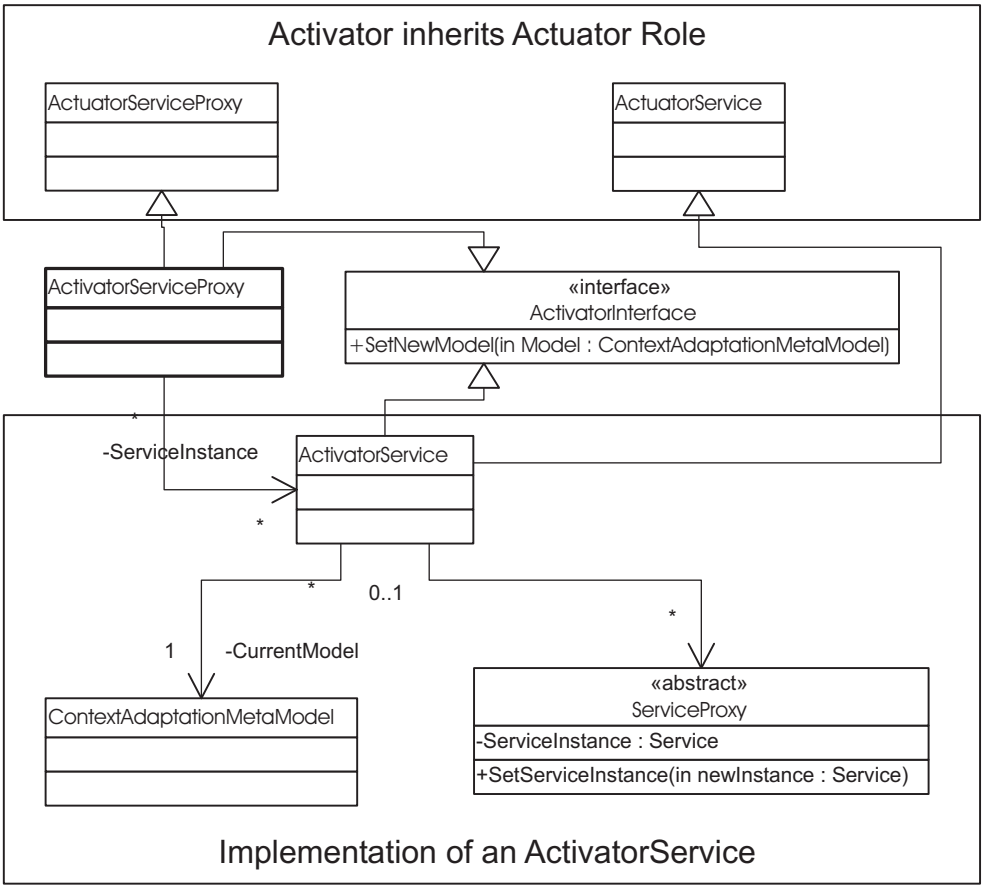


Abb. 34: Der Aktivator ist Aktuator einer 2. Kontextadaption (der Kalibrierung)

Das bedeutet, dass der Aktivator gleichzeitig ein Aktuator sein muss, der mit einem Dienst (`ServiceProxy`) eines Kontextelementes verbunden ist, welches das Metamodell aus Abb. 28 enthält. In diesem Metamodell muss die Kalibrierung als Teil der dort spezifizierten Kontextadaption integriert sein

(d.h., es ist dort beispielsweise ein Modellaktuator beschrieben). Die Spezifikation selbst stammt beispielsweise von einem (ebenfalls darin beschriebenen) Sensor in der Form einer direkten und bewusst zu bedienenden Benutzerschnittelle.

Die Kalibrierung kann so als eine separate zweite Kontextadaption des Teilsystems (A) der ersten Kontextadaption beschrieben werden, die wiederum das Teilsystem (B) der Anwendungsfunktionen adaptiert (Abb. 33). Da der Aktivator (Funktionsbezeichnung) als Aktuator (Dienstbezeichnung) realisiert wird, ist er ein Serviceproxy, der sich jederzeit selbst reprogrammieren kann, indem er seine eigene Komponentenbindung gemäß der im Metamodell enthaltenen Beschreibung verändert. Der Aktivator darf sich allerdings nicht selbst deaktivieren, da dies der unwiderruflichen Entfernung der Kalibrierbarkeit entspräche (Abb. 34).

Die Integration der Kalibrierung in die im Metamodell gespeicherte Spezifikation einer Kontextadaption erfolgt im Framework durch die Definition von Invarianten, welche die Existenz einer Kalibrierung garantieren. Eine Instanz des Metamodells muss also immer ein ausgezeichnetes Kontextelement enthalten, welches das Metamodell selbst enthält und weiterhin mindestens einen ausgezeichneten Aktuator, der den Aktivator darstellt.

Dies ist im Übrigen keine Einschränkung der in 4.1 geforderten vollständigen Reprogrammierbarkeit der Kontextadaption, da sich diese Invarianten als Eigenschaft des Kernsystems ausdrücken lassen (für das keine vollständige Reprogrammierbarkeit gefordert war). Gleiches gilt für die invarianten Kommunikationsbeziehungen zwischen Sensoren, Interpretern, Aktuatoren und Kontextelementen. Diese sind, bezogen auf das Teilsystem der Kontextadaption, ebenfalls nur extrinsische Bedeutungsinvarianten, die ihren Ausdruck in Form von Komponenten des Kernsystems finden und über Serviceproxies syntaktisch abhängig von der Realisierung dieser Komponenten in das Teilsystem der Kontextadaption abgebildet werden. Die Invarianten können daher aufseiten des Kernsystems modelliert werden, wo sie eine Reprogrammierbarkeit der Kontextadaption nicht behindern.

Migration zwischen verschiedenen AdaptionsFramework zur Laufzeit ist möglich (totale Reprogrammierbarkeit).

Das Ganze kann man sich auch anschaulich klar machen. Aufgrund der Entkoppelungseigenschaften von Kontextadaption ist es ohne weiteres möglich, eine kontextadaptive Anwendung zur Laufzeit von einem Framework auf ein anderes zu migrieren. Dazu muss lediglich (z.B. über die Formalisierung aus 5.3) sichergestellt sein, dass beide Frameworks konzeptionell in ähnlicher Weise funktionieren. Die syntaktischen Schnittstellen zwischen den abstrakten Elementen können sich dabei jedoch durchaus unterscheiden. Für die Migration braucht lediglich das Kernsystem partiell rekonfiguriert zu werden, indem die Komponenten des einen Framework nach und nach durch die des

anderen ersetzt werden. Daneben wird über die Kalibrierung das Teilsystem der Kontextadaption vollständig rekonfiguriert, indem zunächst noch durch den alten Aktivator alle alten Service-Proxies deaktiviert werden. Als letzte Tätigkeit übergibt der alte Aktivator an den neuen und deaktiviert sich selbst. Der neue Aktivator aktiviert dann die zu den neuen Framework-Komponenten gehörenden Service-Proxies. Diese können wieder untereinander über eine völlig neue Schnittstelle kommunizieren. Lediglich der neue Aktivator muss den Migrationsprozess explizit unterstützten, indem er sich sowohl als Service-Proxy des alten wie auch des neuen Framework ansprechen lässt.

6.1.5 Minimales initiales System, Bootaktuator und SystemSeed

Ubiquitäre Anwendungen waren so definiert worden, dass sie mithilfe der Kontextadaption ihre Struktur, Implementierung und sogar Funktionalität verändern können. Zu Beginn seiner Initialisierung hat ein solches ubiquitäres System aber immer einen initialen Zustand (Initialsystem).

Im Framework wird dieser Initialzustand durch eine einzelne Komponente (SystemSeed) in einer Art von Bootvorgang hergestellt. Der SystemSeed ist eine (nichtadaptive) Anwendungsfunktion und gehört daher zum Kernsystem einer ubiquitären Anwendung.

Der Initialisierungsprozess des Frameworks ist die Kalibrierung einer Standardadaption.

Zunächst instanziert der SystemSeed den Dienst eines Kontextelementes (ContextServiceProxy) und bindet ihn an eine neue oder existierende Instanz einer geeigneten Komponente (ContextService). In der Regel ist das die zum Framework gehörende Standardimplementierung aller Kontextelemente (ContextServer). Das Kontextelement wird im Anschluss mit einer Spezifikation der initialen Kontextadaption (Spezifikation eines K-Modells nach dem Metamodell von Abb. 28) belegt. Diese Spezifikation kann im SystemSeed als Konstante gespeichert, aus einer Datei oder von einem Webserver geladen worden sein. Der SystemSeed agiert also auch als ein Sensor, der zum Start einer Anwendung deren Spezifikation der initialen Kontextadaption erfasst. Am Ende instanziert der SystemSeed einen Aktuator (Bootaktuator), verbindet diesen mit dem Kontext des Metamodells und beendet seine Tätigkeit.

Bei dem Aktuator handelt es sich um eine Aktivatorkomponente (Abb. 34). Für diesen auch *Modellaktuator* genannten Dienst ist im Framework ebenfalls eine Standardimplementierung vorhanden. Diese erhält das Metamodell der Kontextadaption aus dem Kontext und erzeugt daraus das Teilsystem der Kontextadaption noch einmal neu. Falls gewünscht kann dieser Bootaktivator die vom SystemSeed für den Bootvorgang instanzierte Kontextadaption dabei auch überschreiben, also auch sich selbst durch eine besser für die Anwendung geeignete Implementierung ersetzen.

SystemSeed, ContextServer und BootActivator bilden zusammen mit den beiden Serviceproxies für das Kontextelement und den Bootaktuator das *minimale System* einer ubiquitären Anwendung. Die beiden Proxies gehören dabei zum Teilsystem der Kontextadaption (K-Modell), die restlichen Komponenten, also SystemSeed und die beiden Dienstimplementierungen bilden das Teilsystem der Anwendungsfunktionen (Systemkern). Alle weiteren Anwendungsfunktionen könnten theoretisch durch die initiale Kontextadaption zur Laufzeit aus lokal verfügbaren Ressourcen der Umgebung gebunden werden, oder dem Systemkern durch Rekonfiguration hinzugefügt werden. In der Regel enthält der Systemkern einer Anwendung aber bereits eine Reihe von Funktionen in Form von Komponenten des Framework, um auch unabhängig von externen lokalen Ressourcen eine gewisse Mindestfunktionalität aufrechterhalten zu können.

6.2 Unterstützungsfunktionen für Adaption

In 3.2.2 war Adaption als Fähigkeit, Realisierung und Verhalten an die jeweilige Situation der Anwendung anzupassen bezeichnet worden. In der Formalisierung 5.3 wurde dies aufgrund der einfachen Natur des Basismodells auf die simple Möglichkeit abstrahiert, Komponenten und Kanäle hinzuzufügen oder zu entfernen (genauer gesagt dazwischen zu wechseln).

In der Betrachtungsweise eines technischen (meist objektorientierten) Systemmodells kommen gegenüber der abstrakteren formalen Betrachtungsweise zunächst noch drei weitere Freiheitsgrade hinzu:

- Kontextinformationen lassen sich ohne weitere Interpretation direkt im Kernsystem verwenden (*Contextual Sensing*[46] siehe auch 5.1.3).
- Nachrichten auf Kanälen (z.B. Methodenaufrufe) lassen sich erzeugen (*Context-Triggered Actions* [30] siehe auch 5.1.3).
- Die Architektur des Kernsystems lässt sich in ähnlicher Weise wie bei der Kalibrierung beeinflussen. (*Automatic Contextual Reconfiguration* [30] siehe auch 5.1.3)

Formal lassen sich diese Fälle natürlich wieder auf die ursprünglichen Abstraktionen zurückführen. So lassen sich beispielsweise Methodenaufrufe ganz einfach durch die kurzzeitige Aktivierung einer Komponente abbilden, welche einen beständigen Strom der betreffenden Nachricht am Eingabekanal der Zielkomponente erzeugt.

Die technische Realisierung solcher Formalisierungen ist jedoch teilweise sehr komplex. Das Framework stellt aus diesem Grunde drei weitere Hilfs-

komponenten zur Verfügung, welche die betreffenden Umsetzungen bereits enthalten:

- *DataActuator* ist eine Komponente, die einen generischen Datenzugriff auf Kontextinformationen erlaubt. Die Komponente übernimmt im Wesentlichen das Ein- und Auspacken der Kontextinformationen in und aus ihren Metainformationen. Dies ist notwendig, da die Kontextinformationen in Kontextelementen gleichzeitig als ihr eigenes Metamodell (selbstbeschreibendes Datenformat) fungieren müssen. Diese Konstruktion ist notwendig, um die geforderte Entkoppelungseigenschaft zu erreichen (Genaueres dazu im nächsten Abschnitt).
- *TransactionActuator* ist eine Spezialform eines Aktivators, mithilfe derer dynamisch beliebige Methodenaufrufe für instanzierte Komponenten erzeugt werden können. Die Hilfskomponente stützt sich dabei auf eine in den meisten objektorientierten Laufzeitumgebungen bereits enthaltene Funktionalität (Classloader o.Ä.).
- Der *CoreActuator* entspricht dem TransactionActuator, mit dem Unterschied, dass auch Konstruktor- respektive Destruktormethoden aufgerufen werden und damit neue Instanzen von Komponenten erzeugt oder vernichtet werden können.

Mithilfe der letzten Klasse lässt sich auch das Kernsystem vollständig reprogrammieren. Dies ist aber nur in sehr speziellen Fällen wie etwa der zuvor beschriebenen Migration einer Anwendung zwischen zwei Frameworks wirklich notwendig. Die Möglichkeit hat aber auch noch eine prinzipielle Bedeutung. Sie zeigt nämlich, dass bei richtiger Wahl der Adaption die manuelle Reprogrammierung (Kalibrierung) des Teilsystems der Adaption ausreichen kann, indirekt das Gesamtsystem vollständig zu reprogrammieren. Die Anforderung dazu ergibt sich natürlich aus der gewünschten Kompensation des Frame-Problems und der Tatsache, dass die genaue Einteilung einer ubiquitären Anwendung in die Teilsysteme Kontextadaption und Kernsystem beziehungsweise Umgebung eine Entwurfsentscheidung ist und nicht an allgemein gültigen formalen Kriterien festgemacht werden kann. Dies wiederum bedeutet, dass sich zwar alle vorhersehbar FP-kritischen Interpretationen im Teilsystem der Kontextadaption konzentrieren lassen, jedoch natürlich dennoch FP-anfällige Interpretationen im Kernsystem verbleiben können (jedes Modell ist prinzipiell FP-anfällig). Die indirekte vollständige Reprogrammierbarkeit auch des Kernsystems stellt damit sicher, dass auch seltenere SUBs kalibriert werden können, deren Ursprung auf das Kernsystem zurückzuführen ist. Ein leeres Kernsystem erfüllt im Übrigen den gleichen Zweck.

6.3 Beschreibungstechnik für Adaptionsmodelle

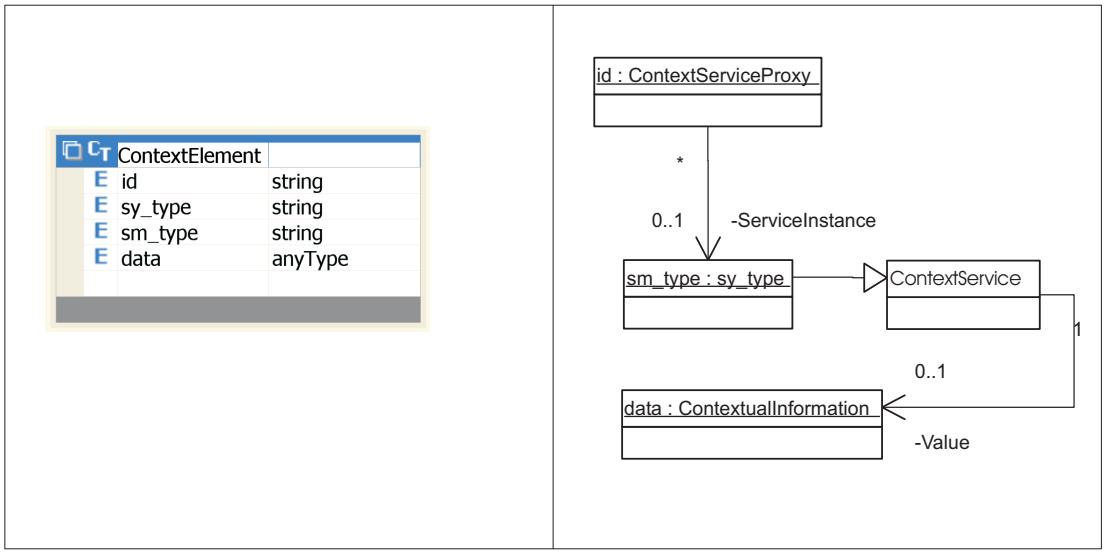
Durch das Metamodell in Abb. 28, welches über die Kalibrierungsschnittstelle in Form des Aktivators akzeptiert wird, ist bereits die Syntax einer Beschreibungstechnik für K-Modelle und damit der Kontextadaption gegeben. Die Semantik einer solchen Beschreibung lässt sich durch Abbildung auf die formale Definition (siehe 5.3) der einzelnen Elemente herstellen.

Da für die Realisierung der K-Modelle Belegungen des Metamodells in einem Kontextelement gespeichert werden müssen und diese nur Daten in Form von selbstbeschreibenden Dokumenten speichern, wird die durch das Metamodell in Abb. 28 skizzierte Beschreibungssprache für die Kalibrierung im Folgenden als XML Dokumententyp definiert.

6.3.1 Kontextelemente

Für die Definition einer Beschreibungssprache für K-Modell gemäß dem in Abb. 28 gezeigten Metamodell, lassen sich dessen einzelne Elemente auf diese Weise in Form einer XML-Schemadefinition spezifizieren und auf die im Framework gegebene Realisierung des in 5.3 formal spezifizierten Modellelementes abbilden.

ContextElement



Der XSD Typ *ContextElement* (links) ist dabei wie folgt definiert:

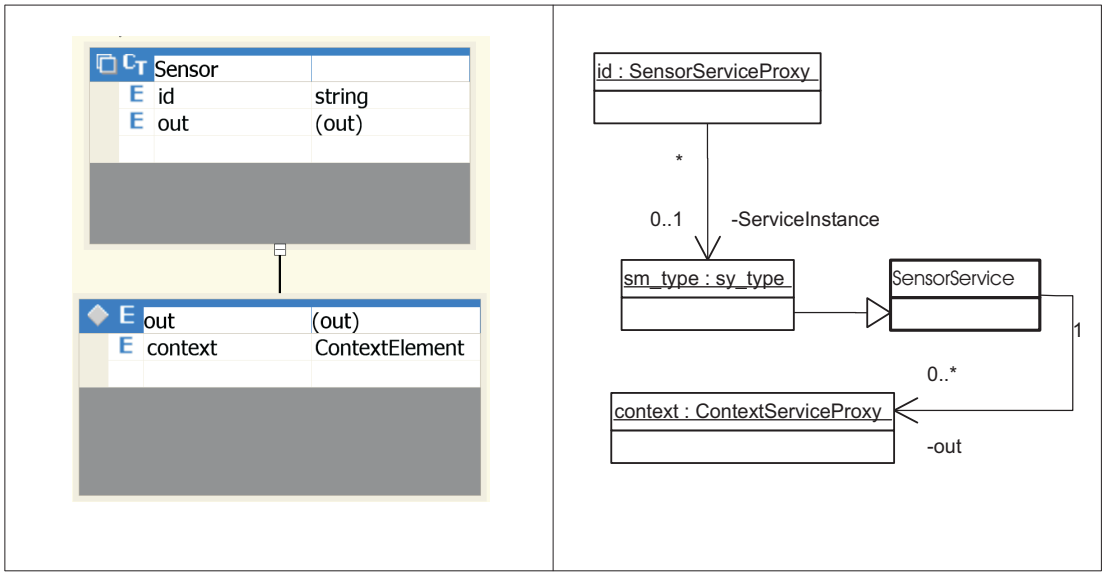
```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="ContextElementSchema">
 <xs:complexType name="ContextElement">
  <xs:sequence>
   <xs:element name="id" type="xs:string" />
   <xs:element name="sy_type" type="xs:string" />
   <xs:element name="sm_type" type="xs:string" />
   <xs:element name="data" type="xs:anyType" />
  </xs:sequence>
  <xs:attribute name="group" type="xs:string" />
  <xs:attribute name="category" type="xs:string" />
 </xs:complexType>
</xs:schema>
```

In der Abbildung wird auch noch einmal der Unterschied zwischen semantischem und syntaktischem Typ klar. Während der syntaktische Typ eine bestimmte Art der Datenstruktur als Interpretation der untypisierten aber sich selbst beschreibenden Kontextinformationen darstellt, identifiziert der semantische Typ im Fall der Kontextelemente eine bestimmte Bedeutung (konkrete Instanz). In der Regel wird die gezeigte Abbildung für Kontextelemente im Framework jedoch vereinfacht. Dazu war der Dienst ContextService im Framework bereits so konstruiert worden, dass er auch einen gemeinsamen Zugriffspunkt für eine Menge von Kontextelementen des gleichen syntaktischen Typs zur Verfügung stellen kann.

Die beiden Attribute *group* und *category* stellen lediglich Strukturierungsmerkmale der Beschreibungstechnik dar und übertragen sich nicht formal in die Realisierung. Sie dienen lediglich als Hilfestellung für die Definition von syntaktischen Transformationen der Beschreibungstechnik, beispielsweise um Elemente höherer Ordnung auszudrücken, die sich aus mehreren Grundelementen zusammensetzen.

6.3.2 Sensoren

SensorElement



Der XSD Typ *Sensor* ist wie folgt definiert:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="SensorSchema" >
 <xs:complexType name="Sensor">
  <xs:sequence>
   <xs:element name="id" type="xs:string" />
   <xs:element name="sy_type" type="xs:string" />
   <xs:element name="sm_type" type="xs:string" />
   <xs:element name="out">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="context" type="xs:ContextElement" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="group" type="xs:string" />
  <xs:attribute name="category" type="xs:string" />
 </xs:complexType>
</xs:schema>
```

Er beschreibt eine bestimmte Realisierung des in 5.3 formal spezifizierten Sensorelementes innerhalb des Framework.

`Sy_type` und `sm_type` können, da es sich ja um Teile eine Kontextinformation handelt, wiederum beliebige Dokumente sein, solange die anderen K-Modellelemente, die darauf Zugriff nehmen, diese Beschreibung auch richtig interpretieren können.

Im obigen Fall müssen also bestimmte Komponentenrealisierung des Framework den Metamodellkontext jedes Sensorelementes dahingehend interpretieren können, dass sie dazu in der Lage sind, zur Laufzeit eine Sensorkomponente an die Beschreibung zu binden.

So enthält das Framework beispielsweise einen Modellaktuator (d.h. Aktivator-Komponente, die den Aktuator Dienst erfüllt), welcher `sy_type` Beschreibungen als Verweise auf WSDL-Dokumente und `sm_type` entweder als einfache Ontologie (namespace) oder als die URL einer konkreten Webservicekomponente interpretieren kann. Im Fall der Ontologie wird diese über einen Verzeichnisdienst lokal verfügbarer Umgebungskomponenten (Discovery Service) ebenfalls auf eine URL abgebildet.

Zunächst wird auf diese Weise die `out` Referenz des mit dem Sensor verbundenen Kontextelementes aufgelöst und dann die damit referenzierte Komponente an den entsprechenden `ContextServiceProxy` über die Methode `SetServiceInstance` gebunden. In der Regel ist das der im Framework enthaltene Kontextserver.

Auf die gleiche Weise wird auch die Sensorreferenz selbst aufgelöst und an eine Dienstabhängigkeit des Kontextservers gebunden, falls es sich um einen Sensor handelt, der im *Poll*-Verfahren abgefragt wird. Im Falle dass der Sensor seine Informationen per *Push* überträgt, existiert in der Sensorkomponente selbst eine Dienstabhängigkeit zu einem Kontextelement, die an den Kontextserver gebunden wird.

6.3.3 Interpreter und Aktuatoren

Die Abbildung der XSD-Typen *Interpreter* und *Actuator* funktioniert analog zu den Sensoren. Die zugehörigen Schemata sind wie folgt definiert:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="InterpreterSchema">
 <xs:complexType name="Interpreter">
  <xs:sequence>
   <xs:element name="id" type="xs:string" />
   <xs:element name="sy_type" type="xs:string" />
   <xs:element name="sm_type" type="xs:string" />
   <xs:element name="in">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="context" type="xs:ContextElement" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="out">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="context" type="xs:ContextElement" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="group" type="xs:string" />
  <xs:attribute name="category" type="xs:string" />
 </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="ActuatorSchema" >
 <xs:complexType name="Actuator">
  <xs:sequence>
   <xs:element name="id" type="xs:string" />
   <xs:element name="sy_type" type="xs:string" />
   <xs:element name="sm_type" type="xs:string" />
   <xs:element name="in">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="context" type="xs:ContextElement" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="group" type="xs:string" />
  <xs:attribute name="category" type="xs:string" />
 </xs:complexType>
</xs:schema>
```

6.4 Schnittstellen der Kalibrierung

Eng mit der technischen Realisierung einer kalibrierbaren Kontextadaption verbunden ist auch die Implementierung einer geeigneten Benutzerschnittstelle der Kalibrierung (siehe auch tertiäre Interaktion in 3.3.4-Abb. 13 und 4.3). Gerade im Zusammenhang mit ubiquitären Anwendungen bedeutet *ge-*

eignet aber auch, dass Konzeption und Realisierung sowohl von der Art der Anwendung, als auch der Situation der Anwendung selbst abhängen können. Mit anderen Worten; die Benutzerschnittstelle der Kalibrierung ist selbst wieder kontextadaptiv.

> Einem ungeübten Benutzer möchte man zu Anfang vielleicht weniger Kalibrierungsmöglichkeiten bieten, als einem erfahrenen Benutzer. Für einen Systemadministrator oder sogar einen Entwickler kann eine skriptbasierte Kalibrierung geeignet sein, während ein Endbenutzer eine grafische Oberfläche vorzieht. Eine solche kann aber ungeeignet sein, wenn das zu kalibrierende System oder das für die Kalibrierung verwendete Zugangsgerät ein Mobiltelefon ist.

Die genaue Art der Benutzerschnittstelle ist jedoch zu anwendungsspezifisch, um sie in einem Framework zu generalisieren. Das Framework stellt daher nur eine generelle Kalibrierungsschnittstelle und ein geeignetes Austauschformat bereit. Beides kann dann, wie im nächsten Kapitel beschrieben, im Entwurf konkreter Anwendungen zu einer Benutzerschnittstelle verfeinert werden.

6.4.1 Ausdehnung der Kalibrierungsschnittstelle

Die erste Hilfestellung im Entwurf von Kalibrierungsschnittstellen durch das Framework besteht darin, die ursprünglich sehr kompakte Kalibrierungsschnittstelle (ein Kontextelement als Realisierung des Metamodells) auf mehrere Kontextelemente zu verteilen. Dies erlaubt zu einem späteren Zeitpunkt eine einfachere Definition von Benutzerschnittstellen der Kalibrierung, die beispielsweise nur einen einzigen Sensor verändern können sollen. Ebenso lässt sich auf diese Weise die Kalibrierung auf bestimmte Elemente oder Elementtypen beschränken. Dies schränkt natürlich die Kompensationsfähigkeiten für SUB-Phänomene ein, kann aber notwendig sein, um bestimmte Sicherheitseigenschaften (Security) zu garantieren. Und Sicherheit bedeutet nichts anderes, als illegale Situationen von der Nutzbarkeit einer Anwendung auszuschließen.

Die angesprochene Verteilung der Kalibrierungsschnittstelle im Kontext übernehmen die folgenden im Framework definierten Interpreterdienste:

- Der *ModelInterpreter* ist eine Implementierung des Interpreterdienstes, welche aus vier Kontextelementen (SensorModel, InterpreterModel, ActuatorModel und ContextModel) die integrierte K-Modellspezifikati-

on gemäß dem Metamodell erzeugt und in dem Kontext ablegt, der durch den Aktivator (auch ModelActuator genannt) ausgelesen wird.
- Analog erzeugen die Interpreter *SensorModelIp*, *InterpreterModelIp*, *ActuatorModelIp* und *ContextElementModelIp* aus einer Menge von einzelnen Kontextelementen, die jeweils die Spezifikation eines einzelnen Elementes enthalten, ein integriertes Modell für jeden Elementtyp und speichert diese in den vier Kontextelementen, die dem ModelInterpreter als Eingabe dienen.

Für jedes Element des Metamodells existiert also ein eigenes Kontextelement, das die Spezifikation des ersteren Elementes enthält. Eine Ausnahme davon sind natürlich die Kontextelemente. Ansonsten müsste für jedes Kontextelement ein Kontextelement existieren, das dieses beschreibt und für dieses müsste wiederum ein Kontextelement existieren usw. Kontextelemente beschreiben sich aus diesem Grund selbst. Das bedeutet schlicht, dass jedes Kontextelement immer ein XML-Dokument mit seiner eigenen Spezifikation speichert. Diese Spezifikation umfasst wie in 6.3.1 definiert neben den Metadaten auch den (eigentlichen) eigenen Inhalt in Form eines XML Unterbaumes, der nur an der Stelle der Inhaltsspezifikation einkopiert wird, sobald sich der Kontext verändert. Die Nutzdaten werden also von der Metamodellspezifikation eines jeden Kontextelementes umschlossen.

6.4.2 Grafische Benutzerschnittstelle für Kalibrierung

Als zweite Hilfsfunktion enthält das Framework die Spezifikation einer grafischen Benutzerschnittstelle für die Kalibrierung. Der Grund dafür ist, dass dies eine häufig verwendete Form der Kalibrierung darstellt. Das Framework definiert zu diesem Zweck eine Reihe von grafischen Grundsymbolen, die auf die Beschreibungssprache aus 6.3 und die in 6.4.1 beschriebene Ausdehnung der Kalibrierungsschnittstelle abgebildet werden. Wegen letzterem handelt es sich streng genommen um eine grafische Repräsentation einer (Teil-)Benutzerschnittstelle der Kalibrierung, und nicht nur um eine reine grafische Notation der Beschreibungssprache für Kontextadaption. Für die Komplettierung dieser Teilbenutzerschnittstelle müssen spezifische ubiquitäre Anwendungen dann lediglich noch deren genaues Design (Rendering) und die Manipulierbarkeit (Editing) festlegen.

Dennoch eignet sich diese grafische Repräsentation auch als Spezifikationstechnik und wird im weiteren Verlauf der Arbeit zum Zwecke der Veranschaulichung von modelliertem Adaptionsverhalten verwendet. Der Grund dafür ist, dass in 6.4.1 die Ausdehnung der Kalibrierungsschnittstelle als Kontextadaption definiert wurde. Diese kann aber in der Beschreibungssprache

aus 6.3 ausgedrückt werden. Die Transformation der grafischen Notation braucht also lediglich einige Makros enthalten, die sich auf die Implementierung des Framework beziehen. Abb. 35 zeigt die eine Übersicht der grafischen Beschreibung.

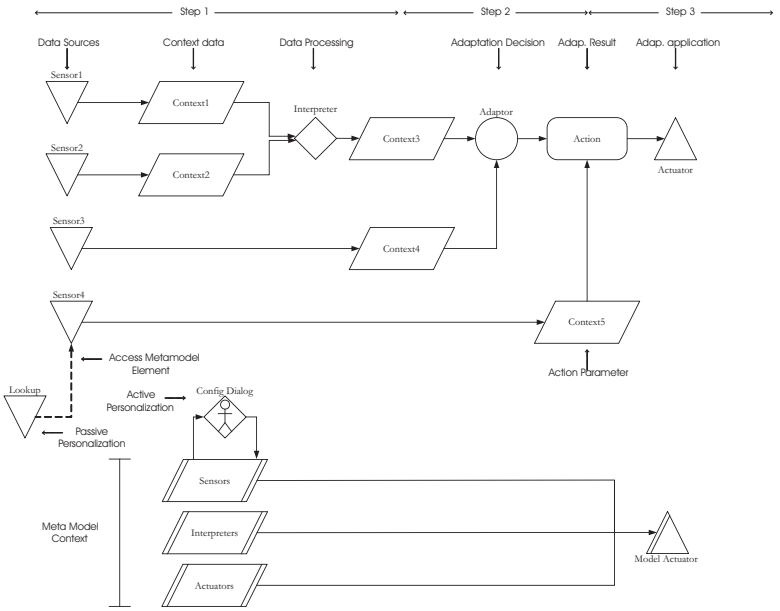


Abb. 35: Graphische Notation im Überblick

Darstellung der Sensoren

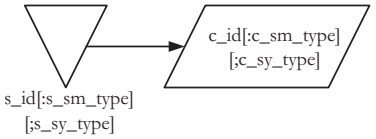
Sensor

 Id[:sm_type] [;sy_type]	<?xml version="1.0" encoding="UTF-8"?> <context> <id>model.sensor.id</id> <sy_type>context.xsd</sy_type> <sm_type>model.sensors.sensor</sm_type> <data> <sensor> <id>id</id> <sy_type>sy_type</sy_type> <sm_type>sm_type</sm_type> <out/> </sensor> </data> </context>

Sämtliche grafischen Elemente werden auf ein Kontextelement abgebildet, das als Inhalt die Spezifikation des dargestellten Elementes in Form der in 6.3 definierten Beschreibungssprache enthält. Diese Kontextelemente gehören zu der in 6.4.1 beschriebenen Ausdehnung des Metamodells der Kontexta-

daption auf einzelne Elemente und sind letztendlich mit einem Modellaktuator (Aktivator siehe 6.1.3) verbunden, welcher die Inhalte der Kontextelemente gemäß den Abbildungen in 6.3 in Dienst-Komponentenbindungen übersetzt und an die Service-Proxies übermittelt.

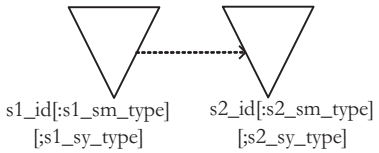
Sensor-Metamodellkontext




```
<?xml version="1.0" encoding="UTF-8"?>
<context>
 <id>model.sensor.s_id</id>
 <sy_type>context.xsd</sy_type>
 <sm_type>model.sensors.sensor</sm_type>
 <data>
  <sensor>
   <id>s_id</id>
   <sy_type>s_sy_type</sy_type>
   <sm_type>s_sm_type</sm_type>
   <out>
    <context>
     <id>c_id</id>
     <sy_type>c_sy_type</sy_type>
     <sm_type>c_sm_type</sm_type>
    <context>
   </out>
  </sensor>
 </data>
</context>
```

Eine durchgezogene Kante zwischen einem Sensor Knoten und einem Kontext Knoten wird einfach in eine entsprechende *out* Referenz der Sensorbeschreibung übersetzt.

Sensor-MetaSensor




```
<?xml version="1.0" encoding="UTF-8"?>
<context>
 <id>model.sensor.s_id</id>
 <sy_type>context.xsd</sy_type>
 <sm_type>model.sensors.sensor</sm_type>
 <data>
  <sensor>
   <id>s1_id</id>
   <sy_type>s1_sy_type</sy_type>
   <sm_type>s1_sm_type</sm_type>
   <out>
    <context>
      <id>model.sensor.s2_id</id>
      <sm_type>model.sensors.
                sensor</sm_type>
   <context>
   </out>
  </sensor>
 </data>
</context>
```

Eine gestrichelte Kante zwischen einem Sensor Knoten und einem beliebigen anderen Knoten wird dagegen in eine `out` Referenz des entsprechenden Meta-Kontextelementes des Knotens, hier am Beispiel eines anderen Sensors, übersetzt. Lesezugriffe (`in` Referenzen) auf Metamodellelemente können dagegen als normale Kontextreferenzen (durchgezogene Kante) symbolisiert werden. Die Eindeutigkeit wird dadurch gewahrt, dass Anfangs- und Endpunkt ein Metamodellelement für einen Sensor, Interpreter oder Aktuator ist, wohingegen normale Lesezugriffe immer nur genau ein Nichtmetamodell-Kontextelement mit genau einem Metamodellkontextelement verbinden. Eine Ausnahme währen lediglich die Kontextelemente selbst. Diese sind aber wie in 6.3 festgelegt, immer gleichzeitig ihr eigener Metamodellkontext, was bedeutet, dass Metamodellzugriffe in diesem Fall identisch mit dem Zugriff auf normale Kontextinformationen sind.

Darstellung anderer Elemente

Grafische Symbole für die verbleibenden Elemente Aktuator und Interpreter werden analog zur Darstellung des Sensors definiert.

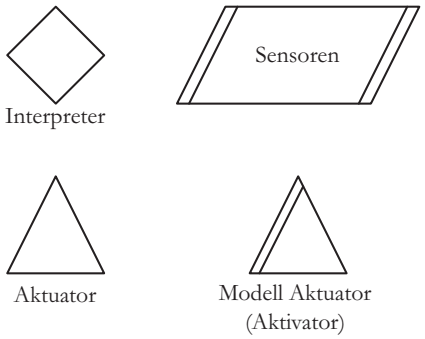


Abb. 36: Weitere Symbole der grafischen Darstellung

Ferner lassen sich mithilfe der Strukturierungsattribute *group* und *category* der in 6.3 definierten Beschreibungstechnik auch grafische Symbole mit strukturierendem oder hierarchisierendem Charakter definieren. Ein Beispiel für ersteres ist die Markierung spezieller Instanzen von Elementen, die im Framework der Verarbeitung des Metamodells dienen mit einem Doppelstrich. Dazu gehören Kontextelemente, welche die einzelnen Metakontextelemente der Sensoren, Interpreter oder Aktuatoren zu größeren Teilmodellen zusammenfassen, beispielsweise die von dem in 6.4.1 beschriebenen Modellinterpreter *SensorModelIp* erzeugte Liste aller Sensoren, aber auch der Modellaktuator (Aktivator).

6.4.3 Strukturierung des Kontextes

Ein anderes Beispiel für die Strukturierung durch Gruppierungsattribute ist die Auszeichnung spezieller Interpreter oder Kontextelemente nach den typischen drei Teilprozessen der Kontextadaption um eine bessere Übersichtlichkeit von grafischen K-Modelldarstellungen zu gewährleisten.

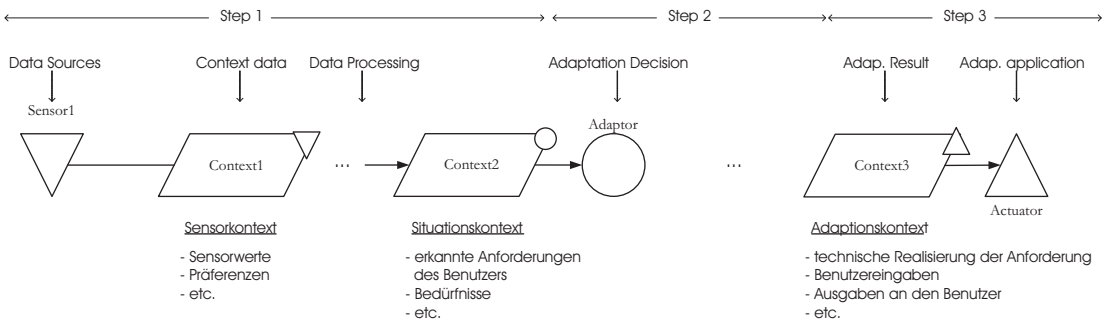


Abb. 37: Strukturierung von Kontextinformationen

Kontextinformationen gelangen nämlich typischerweise über Sensoren in den Kontext des betrachteten Systems. Da nicht immer alle Informationen in der benötigten Art und Weise von Sensoren zu erhalten sind, können sie im Anschluss innerhalb des Kontextes durch Interpreter weiterverarbeitet und verknüpft werden. In einfachen Fällen wird die Information dann direkt über einen Aktuator an das Kernsystem weitergeleitet und dort verwendet (implizite Adaption oder *Context Sensing* genannt). Andernfalls kann mithilfe von speziellen Interpretern (Adaptoren) aus mehreren Einzelinformationen eine bestimmte Situation der Umgebung erkannt werden. Daraus ergeben sich für die Situation gültige Anforderungen oder Bedürfnisse des Benutzers. Diese können direkt von Aktuatoren ausgewertet werden, falls es sich um bekannte Anforderungen handelt, die situationsunabhängig realisiert werden können, beispielsweise durch einfaches Umschalten zwischen zwei Funktionen. Andernfalls kann auch die technische Realisierung im Kontext unter Hinzuziehung weiterer Umgebungsinformationen berechnet werden. Beispiel dafür sind etwa das Entdecken gerade verfügbarer Implementierungen eines Dienstes. Aus dieser Sichtweise heraus können die Kontextelemente einer Kontextadaption vergleichbar den drei Teilprozessen in Gruppen eingeteilt werden (Abb. 37):

- *Sensorkontext* bezeichnet die mit einem Sensor verbundenen Kontextelemente.
- *Situationskontext* bezeichnet mit einem Adaptor verbundene Kontextelemente, die Informationen über erkannte Situationen ausdrücken.
- *Adaptionskontext* bezeichnet das Ergebnis einer Adaptionsentscheidung. Die zum Adaptionskontext gehörenden Kontextelemente sind mit einem Aktuator verbunden.

Die verbleibenden Kontextelemente enthalten im Kontext abgelegte Zwischenergebnisse (*Zwischenkontext/Intermediate Context*).

KAPITEL 7 Methodik

Wie sieht eine methodische Anwendungsentwicklung auf der Basis von K-Modellen aus?

Dieses Kapitel beschreibt eine Methodik für den Entwurf der kalibrierbaren Kontextadaption einer konkreten ubiquitären Anwendung mithilfe des in Kapitel 6 beschriebenen Framework. Die Methodik konzentriert sich dabei auf die Konstruktion des kontextabhängigen Verhaltens sowie seiner Kalibrierungsmöglichkeiten, nicht aber auf den Entwurf der zu adaptierenden Anwendungsfunktionen selbst. Diese können auf herkömmliche Weise als Komponenten entworfen und bereitgestellt werden.

Kapitel 2 beschreibt unter anderem eine ubiquitäre Lichtsteuerung. Je nach Situation kann das Licht automatisch, per Schalter oder über das Smartphone von unterwegs gesteuert werden. Diese Anwendung lässt sich zerlegen in eine nicht FP-kritische Anwendungsfunktion (schaltet das Licht abhängig vom übergebenen Kommando) und die FP-kritische Adaptionslogik (bei Bewegung Schaltfunktion aktivieren, etc.).

Eine solche Aufteilung in anwendungsunspezifische und daher leicht wiederverwendbare Funktionsbausteine sowie eine anwendungsspezifische Kontroll- und Interaktionslogik ist ein relativ häufig angewandtes Strukturierungsmuster in der Softwareentwicklung (Mehrschichten- oder Multi-Tier Architekturen, Trennung in Business Logik und Präsentationsschicht, etc.).

Der erste Abschnitt dieses Kapitels beschreibt analog, anhand welcher Kriterien auch für ubiquitäre Anwendungen eine solche Unterteilung erfolgen kann. In diesem Fall in Kontextadaptionsfunktionen und Anwendungsfunktionen. Der zweite Abschnitt beschäftigt sich mit der Konstruktion einer initialen Kontextadaption, während im dritten Abschnitt der Entwurf der Kalibrierung beschrieben wird.

7.1 Kontext Requirements Engineering

Wichtigste Voraussetzung für die erfolgreiche Kalibrierung von SUB-Phänomenen in einer ubiquitären Anwendung ist die richtige Teilung des Gesamtsystems in ein Teilsystem mit FP-kritischen Adaptionsfunktionen (Kontextadaption) und ein Teilsystem mit FP-unkritischen Anwendungsfunktionen (Kern). Verbleiben nämlich einige der für das Frame-Problem anfälligen Interpretationen (Modellierungen) der Realität im Anwendungskern, können diese möglicherweise nicht ausreichend von der Kalibrierung erfasst werden. Dieser Punkt war bereits in 6.2 im Zusammenhang mit der Migration einer ubiquitären Anwendung zwischen zwei Frameworkimplementierungen diskutiert worden. Als Folge können SUB-Phänomene entstehen, die sich nicht beseitigen lassen. Auf der anderen Seite steigt die Komplexität und sinkt die Vorhersagbarkeit des Systemverhaltens je geringer der Anteil der Kernfunktionen und damit je größer der Anteil der Adaptionen wird. Zudem gibt es keine generell gültigen formalen Kriterien mithilfe derer im Entwurf zwischen Adaption und angepasstem System unterschieden werden könnte, solange dessen Grenzen nicht bereits anderweitig festgelegt sind, etwa durch die Verwendung bereits vorhandener Komponenten. Folglich ist es eine Entwurfsentscheidung, welche Teile einer konkreten ubiquitären Anwendung unter gegebenen Entwicklungsvoraussetzungen (Ressourcen, Anwendungsfall etc.) als FP-kritisch oder unkritisch angenommen werden. Entscheidungskriterien basieren dabei in der Regel auf empirischen Erfahrungswerten, prinzipiellen Überlegungen und methodischen Abschätzungen.

Systemgrenzen in ubiquitären Anwendungen sind eine Entwurfsentscheidung

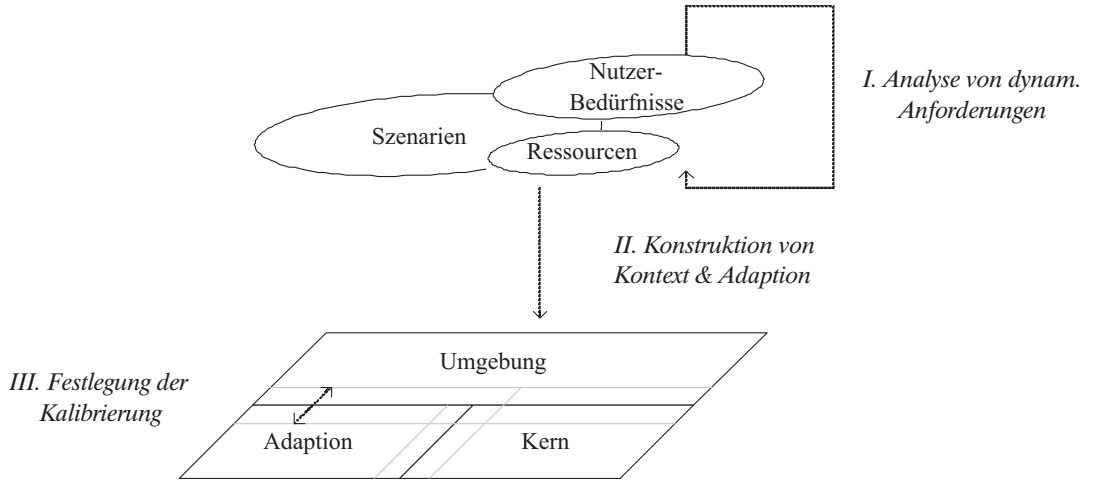


Abb. 38: Entwurfsmethodik für Kontextadaption

Die in diesem Kapitel beschriebene Methodik beschreibt einen solchen Satz von Entscheidungskriterien, die aus den Erfahrungen der Fallstudie und den Modellüberlegungen gewonnen wurden.

Ausgangspunkt ist dabei die Analyse von Nutzerbedürfnissen und deren Abhängigkeit von bestimmten Situationen und Ressourcen analog zu den beiden Hauptdimensionen der Ubiquität (Verfügbarkeit und Nutzbarkeit). Ziel ist dabei zunächst, die dynamischen, d.h. sich ändernden Anforderungen eines zu entwickelnden Systems möglichst vollständig zu ermitteln (Abb. 38).

7.1.1 Zusammenhang zwischen Adaption und Anforderungen

Adaption als Behandlung dynamischer Anforderungen

Ohne den Bezug auf ein technisches Systemmodell kann Adaption nämlich zunächst auch so gedeutet werden, dass ein Teil der definierten Anforderungen (im Allgemeinen Annahmen über die Systemverwendung) bezogen auf die Lebensdauer des Systems keine globale Gültigkeit besitzt, sondern nur in einem bestimmten Kontext.

Definition 20: *Dynamische Anforderungen*

Anforderungen besitzen eine auf die Lebensdauer des Systems bezogene globale Gültigkeit, während dynamische Anforderungen eine auf die Situation des Systems bezogene Gültigkeitsbedingung besitzen.

Trotz dieser klaren Definition ist die tatsächliche Entscheidung, ob in einem geplanten Anwendungssystem dynamische Anforderungen existieren im Allgemeinen schwierig. Der Grund dafür ist, dass Begriffe wie Adaption und Kontext aus einem Umgebungsbegriff heraus definiert werden. Der Umgebungsbegriff ergibt sich wiederum relativ aus einer Systemgrenze. Da für den Begriff des Systems aber keine natürlichen Kriterien der Abgrenzung existieren, sind folglich auch Adaption, Kontext und dynamische Anforderungen eine reine Frage des Betrachtungsstandpunktes.

Man kann allerdings versuchen, diese Grenzen methodisch auszuloten, indem man zunächst von einem idealen ubiquitären System ausgeht. Dieses ideale System erfüllt situationsabhängig alle Nutzerbedürfnisse und alle dafür notwendigen Ressourcen sind immer verfügbar. Alle Nutzeranforderungen werden also zunächst als dynamisch und alle technischen Anforderungen, die zur Umsetzung einer Nutzeranforderungen notwendig sind, als statisch angenommen. Man kann sich das Ganze auch so vorstellen, dass man zunächst von einem gegebenen Kernsystem ausgeht, das alle Benutzerbedürfnisse erfüllen kann und für das im ersten Schritt lediglich ein situationsabhängiger Auswahlmechanismus (die Adaption) konstruiert werden muss, der festlegt,

wie die Bedürfnisse eines Benutzers erkannt werden können (Konstruktion der Nutzbarkeitsdimension der Ubiquität). Im zweiten Schritt wird das im ersten Schritt angenommene Kernsystem realisiert. Dabei werden dann die Grenzen der Realisierbarkeit des Kernsystems berücksichtigt, das heißt, dass manche Bedürfnisse nur unter Einbeziehung der Umgebung realisiert oder in ungünstigen Umgebungssituationen auch gar nicht beziehungsweise nur in schlechter Qualität realisiert werden können. Dafür werden weitere Auswahlmechanismen benötigt, die den Adaptionen der Nutzbarkeitsdimension hinzugefügt werden (Verfügbarkeitsdimension der Ubiquität).

7.1.2 Beginn der Analyse

Ermittlung von dynamischen Anforderungen und ihren Gültigkeitsbedingungen

Zunächst aber beginnt die Analyse wie bereits skizziert mit der Ermittlung der Nutzerbedürfnisse, die in Form von dynamischen Anforderungen spezifiziert werden. Aufgrund des im letzten Abschnitt beschriebenen Zusammenhangs zwischen dynamischen Anforderungen und Adaption, lassen sich diese bereits mithilfe der Beschreibungstechnik aus 6.3 in Form eines Grobentwurfs der Kontextadaption auf Basis des in Kapitel 6 definierten Framework modellieren. Die Gültigkeitsbedingungen der Anforderungen werden dabei als Interpretationen abgebildet. Da es sich oft um sehr allgemeine Interpretationen handelt, die aus einem noch nicht näher spezifizierten Kontext unmittelbar eine Adaptionsentscheidung in Form der Erfüllung eines Nutzerbedürfnisses ableiten (Situationskontext siehe 6.4.3), werden diese Interpretationen mit dem Strukturierungsmerkmal des Situationsadaptors versehen.

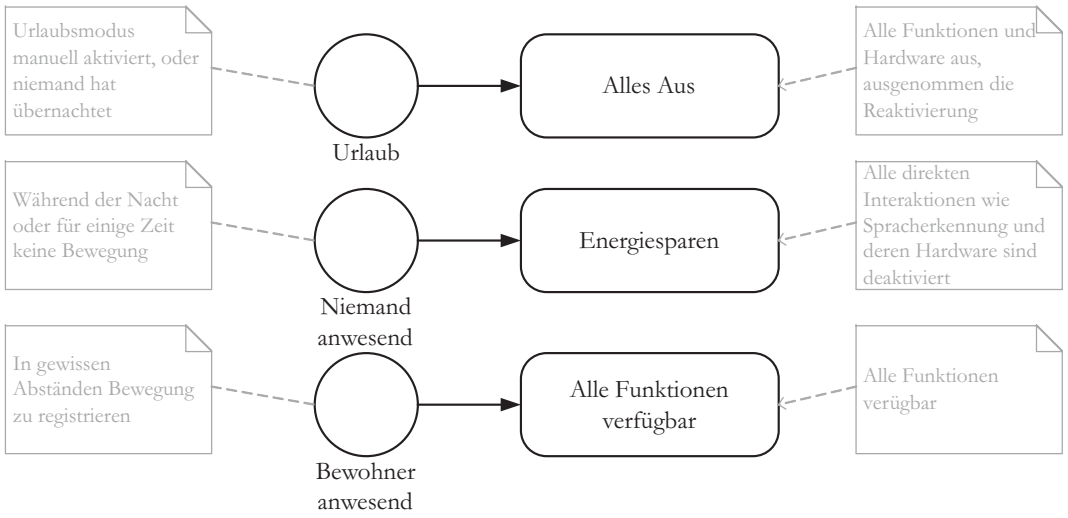


Abb. 39: Beispiel dynamischer Anforderungen

Auf diese Weise können sie später leicht der Übersichtlichkeit halber von den technisch geprägten Interpretationen unterschieden werden. Eine genauere Beschreibung der Strukturierungselemente *Situationsadaptor* und *Aktion* findet sich später in 7.2.2. Für den Moment ist es jedoch ausreichend, die beiden Elemente einfach als spezialisierte Interpreter respektive Aktuatoren anzusehen.

Das Beispiel in Abb. 39 zeigt drei dynamische Anforderungen aus der Fallstudie in Kapitel 2, die aus den Nutzerbedürfnissen in Zusammenhang mit Anwesen- oder Abwesenheit von Personen im Bereich des Wohnraumes gewonnen wurden. Die Anforderungen werden als Adaptionsaktionen ausgedrückt, die Gültigkeitsbedingungen als Situationen, wobei zunächst nur Textfragmente in Form von Kommentaren die einzelnen Elemente genauer spezifizieren.

7.1.3 Verfeinerungsschritte

Die Darstellung der zuvor beschriebenen dynamischen Anforderungen einer Adaption als K-Modell hat schon in der Ausgangslage den Vorteil, dass sich damit Benutzerbedürfnisse in sehr direkter Weise bezüglich eines nicht technisch geprägten Wahrnehmungsmodells darstellen lassen, da die verwendeten Basisbegriffe der Situation und Aktion im K-Modell mit ihrem intuitiven Bedeutungsverständnis übereinstimmen.

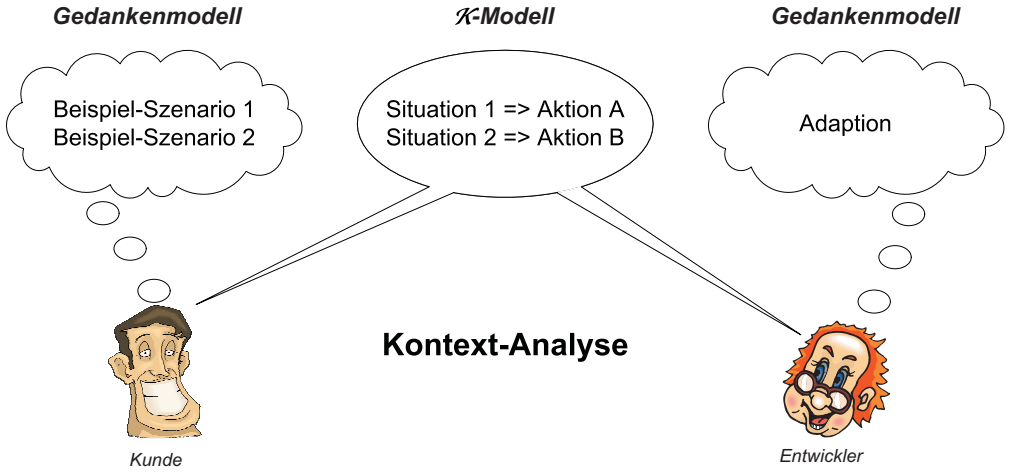


Abb. 40: Kontext Analyse

Gleichzeitig erlaubt die Darstellung als K-Modell die Anwendung einiger einfacher Konstruktionsregeln, welche die Strukturierung und Detaillierung der

zu Anfang möglicherweise ungeordneten Ideensammlung hin zu einer validierbaren Modellierung einer Adaption verbessern können.

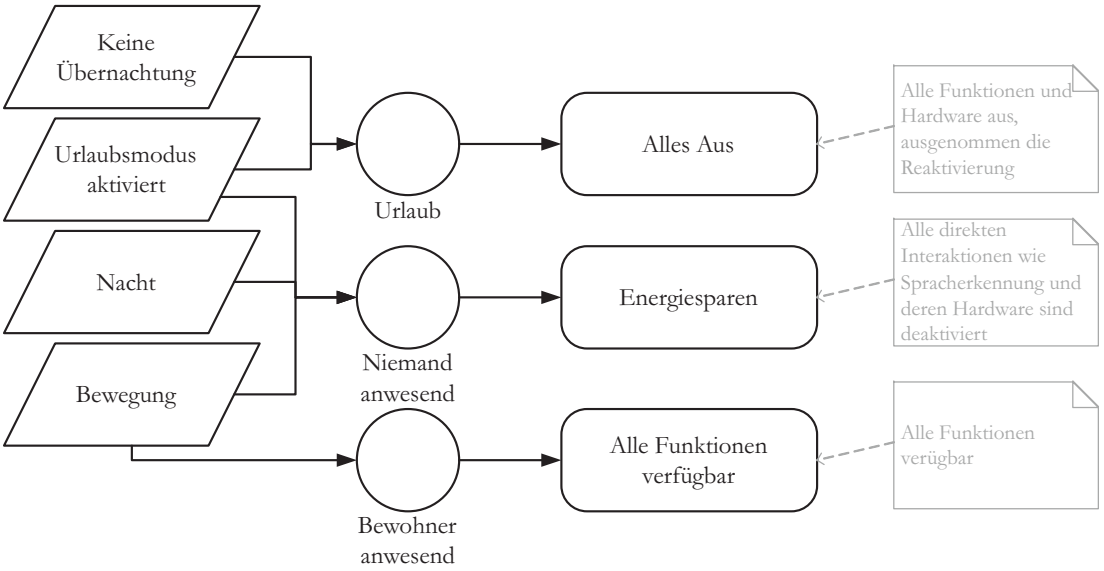


Abb. 41: Beispiel für detailliertere Gültigkeitsbedingungen

Der erste Schritt besteht dabei in einer genaueren Spezifizierung der Situation, also der Gültigkeitsbedingungen der sich hinter den Aktionen verbergenden dynamischen Anforderungen. Dies geschieht durch die Festlegung einzelner Kontextinformationen (siehe Abb. 41), welche zur Erkennung einer Situation notwendig sind. Auf diese Weise können anschließend die folgenden Fragestellungen besser bearbeitet werden:

- Gibt es noch weitere Anforderungen/Aktionen/Bedürfnisse, die in der gleichen Situation gelten sollen?
- Können die Anforderungen noch weiter zerlegt werden?
- Treten umgekehrt einige der Anforderungen auch noch in anderen dargestellten Situationen auf?
- Decken die existierenden Situationen alle Belegungsmöglichkeiten des Situationskontextes ab, oder können noch weitere Situationen unterschieden werden?

Ein wichtiges Hilfsmittel bei der Beantwortung dieser Analysefragen sind die folgenden Metriken.

7.1.4 Metriken

An manchen Stellen der Analyse von Kontextadaption eines konkreten Systems kann es vorteilhaft sein, eine Aussage über die resultierende Adaptions-

fähigkeit eines Softwaresystems machen zu können. Eine solche Aussagemöglichkeit hat nahe liegende Vorteile bei der Beurteilung beispielsweise der möglichen Lebensdauer eines Systems, der wirtschaftlichen Abwägung zwischen Aufwand und Nutzen oder kann Hinweise auf den Grad der im Entwurf explizit zu modellierenden Aktionen, Situationen und Kontexte geben.

Solche und andere Metriken lassen sich leicht auf der Basis des in Kapitel 6 definierten Framework und dem dort verwendeten Metamodell der Kontextadaption definieren, da der Adaptionsvorgang dort klar in drei Schritten definiert und in entsprechende Elemente (Kontexte, Sensoren, Interpreter, Aktuatoren sowie die zusammengesetzten Elemente Adaptoren und Aktionen) der Beschreibungstechnik abgebildet wird. Mit der richtigen Gewichtung versehen lässt sich daraus beispielsweise eine Aussage über die Adaptivität gewinnen.

Adaptivitätsmetrik

Basis der Adaptivität bezüglich des Framework in Kapitel 6 sind die Aktuatoren und die ihnen vorgeschalteten Adaptionskontexte, die zur besseren Erkennung der Adaptionsphasen im Modell als Aktionen markiert werden. Ein Adaptionsmodell ohne Aktionen und Aktuatoren besitzt keine Möglichkeit der Interaktion mit dem Kernsystem, was mit einer Adaptivität von 0 gleichgesetzt werden kann.

Markieren die Aktionen das Adaptivitätspotenzial eines Systems, so lässt sich aus der Anzahl der erkennbaren Situationen in Form von Adaptoren und den direkt verbundenen Initalkontexten eine Granularität der Adaptionsfähigkeit ablesen. Ein Kontextadaptionsmodell ohne Situationen besitzt zwar ein hohes Adaptivitätspotenzial, kann dieses jedoch nicht verwirklichen. Sensoren und die Herleitung des Initialkontextes spielen dabei keine Rolle.

Anders liegt der Fall, wenn die Kalibrierung des Modells in Spiel kommt. Modifizierungen einzelner Elemente sind durch die bisherige Metrik abgedeckt, solange keine neuen Elemente hinzugefügt oder entfernt werden können. Modifizierbare Elemente werden indirekt zu Aktionen (da sie mit Aktuatoren verbunden sind, welche die Modifikationen umsetzen) und sind darüber durch die Metrik abgedeckt, auch wenn es sich um Sensoren oder Kontextelemente zur Berechnung des Initialkontextes handelt. Nicht abgedeckt sind jedoch alle Fälle des Hinzufügens von Elementen in das Modell. Dadurch steigt die potenzielle Adaptivität ins Unendliche. Genauere Aussagen lassen sich darüber nicht machen, da die technischen Details der Logik des Hinzufügens und etwaiger Constraints in den Implementierungen einzelner Elemente verborgen sind. Dies macht den generellen Vergleich zweier Adaptionsmodelle

in solchen Fällen nur für einen bestimmten Kontext möglich. Mit anderen Worten kann sich durch die Kalibrierung die Adaptivität eines Systems natürlich verändern.

Definition 21: *Adaptivität*

$$\sum_{s\ Adap} exclusiv(s)$$

$$exclusiv: Adap \mapsto \mathbb{Q};\ exclusiv(s) \qquad \begin{cases} \dfrac{1}{\left|usedby(actof(s))\right|}; actof(s) \\ 0; sonst \end{cases}$$

$actof: Adap \mapsto \mathbb{P}Act$; von einem Adaptor ausgelöste Aktionen
$usedby: \mathbb{P}Act \mapsto \mathbb{P}Adap$;
$usedby(x)$ $\{s$ $Act: actof(s)$ $x\}$; Adaptoren mit gleichen Aktionen

Für die Berechnung der Metrik wird über alle in Form von Adaptoren, also speziellen Interpretern zur Erkennung einer Situation deren Exklusivität (*exclusiv*) addiert. Die Exklusivität einer Situation richtet sich danach, wie stark deren Aktionen mit anderen Situationen überlappen. Dies wird danach gemessen wie oft die einer Situation zugeordneten Aktionen (*actof(s)*) auch in anderen Situationen verwendet werden (*usedby(x)*). Folglich hätte das Beispiel aus Abb.41 eine Adaptivität von 3.

Die Metrik ist so gewählt, dass ein System ohne Aktionen oder durch Adaptoren erkannte Situationen eine Adaptivität von 0 besitzt. Ein System, das lediglich eine Situation unterscheiden kann besitzt eine Adaptivität von maximal 1. Bei der Bewertung einzelner Adaptoren wird die jeweilige Kombination der ausgelösten Aktionen nach ihrer Exklusivität bewertet. Dies verhindert die Mehrfachwertung einer einzelnen Kombination. Ein System, das lediglich eine einzige Aktion kennt und diese in vielen unterschiedlichen Situationen anwendet, besitzt ebenfalls eine Adaptivität von maximal 1. Die Anzahl der unterscheidbaren Aktionen (Aktionskombinationen) definiert die maximale potenzielle Adaptivität (2^n 1) für den Fall, dass sich keine Aktionen hinzufügen oder entfernen lassen.

7.1.5 Balancierungs-Metriken

Auf der Grundlage der Adaptivitätsmetrik und der für ihre Definition herangezogenen Hilfsfunktionen lassen sich zudem eine Folge von Balancierungs-Metriken definieren, die bei der Anforderungsanalyse von Kontextadaption hilfreich sind:

- Das Verhältnis der Anzahl der von einem Situationsadaptor ausgelösten Aktionen zum Durchschnittswert kann ein Indikator dafür sein, dass der betroffene Situationsinterpreter zu viele implizite Adaptionen verbirgt, die man noch explizit modellieren könnte.
- Eine geringe Exklusivität einer Adaptionsaktion deutet auf eine starke Vernetzung der einzelnen Adaptionsentscheidungen hin. Dies kann darauf hindeuten, dass der wahre Zusammenhang einer Adaptionsentscheidung nicht erkannt und deshalb auf mehrere Einzelsituationen verteilt wurde.
- Eine bezogen auf die anderen Adaptoren eines Systems überdurchschnittlich hohe Anzahl von Elementen eines Situationskontextes (die Kontexte, aus denen eine Situation heraus erkannt wird) kann darauf hindeuten, dass die Adaptionsentscheidung zu stark an technischen Details aufgehängt wurde. Daher könnte eine bessere Strukturierung dieses Kontextes in mehrere Interpretations- und Auswertungsstufen notwendig sein.

7.2 Entwurf der initialen Kontextadaption, des Kernsystems und der Kalibrierung

Nachdem die Analyse-Phase beendet wurde, verfügt man bereits über eine strukturierte Spezifikation der dynamischen Anforderungen und deren jeweiliger Kontexte, welche ihre Gültigkeitsbedingungen beschreiben.

Aus diesem Ergebnis kann im nächsten Schritt nun die konkrete ubiquitäre Anwendung entworfen werden. Gemäß den Vorgaben des Framework aus Kapitel 6 besteht diese aus den zwei Teilsystemen:

- *Kontextadaption*, also die Realisierung eines K-Modells aus Sensor-, Kontext-, Interpreter- und Aktuatordiensten mithilfe des Framework. Die *Kalibrierung* wird dabei (nach 5.1) als Teil der Kontextadaption ausgedrückt.
- *Anwendungskern*, also den Implementierungen oder Diensten der zu adaptierenden Funktionen.

Der Entwurf der beiden Teilsysteme kann dabei nicht streng sequenziell in zwei separaten Phasen erfolgen, da zwischen Kern und Adaption zirkuläre Abhängigkeiten bestehen (siehe auch mathematisches Modell in 5.3). So kön-

nen beispielsweise einige Entwurfsentscheidungen im Anwendungskern dazu führen, dass zusätzliche Adaptionen hinzugefügt werden müssen.

Kernsystemfunktionen können wie die Kontextadaption auch für ihre Realisierung etwa auf externe Softwarekomponenten zurückgreifen, beispielsweise in Form von Dienstabhängigkeiten. Solche Dienste müssen zur Laufzeit an eine externe Komponente gebunden werden. Kann dabei absehbar aus einer größeren Menge von Möglichkeiten gewählt werden und sind die Auswahlkriterien situationsabhängig (Kosten, Qualität, etc.), dann ist es meist sinnvoll, diese technische Adaption ebenfalls als (sekundäre) Kontextadaption zu modellieren (z.B. abhängig von Person, benötigter Funktionalität usw.).

Umgekehrt können Entwurfsentscheidungen im Teilsystem der Kontextadaption dazu führen, dass zusätzliche Kernsystemfunktionen hinzugefügt werden müssen. Auch hier sind meist externe Abhängigkeiten der Grund, also Entscheidungen, ob die Implementierung einer Adaption abgeschlossen bezüglich des Systems der ubiquitären Anwendung ist, oder Abhängigkeiten zu externen Funktionen bestehen.

Gemäß der Definition der K-Modelle und auch den Festlegungen des Framework besteht das Teilsystem der Kontextadaption ausschließlich aus Diensten, die zur Laufzeit an eine Komponente gebunden werden. Diese können (wie bei den meisten Sensoren) aus der Umgebung stammen, oder (wie bei den meisten Aktuatoren) zum Kernsystem gehören. In Ausnahmefällen können sich Adaptionen auch auf Kontexte beziehen, die Informationen über den Zustand des Kernsystems enthalten (z.B. Auslastung, Reaktionsgeschwindigkeit etc.). In solchen Fällen muss beispielsweise im Kernsystem eine Sensorimplementierung hinzugefügt und fest an den Sensordienst in der Kontextadaption gebunden werden.

In der Fallstudie Kapitel 2 hat sich daher aufgrund dieser Zirkularabhängigkeiten gezeigt, dass in der Regel im Anschluss an den Erstentwurf der Adaption in der Analyse ein dreiphasiger Entwurfszyklus (Kernsystem, Adaption, Kernsystem) für die Modellierung der sekundären Verfügbarkeitsadaptionen angebracht ist. Die einzelnen Kriterien, nach denen solche sekundären Adaptionen hinzugefügt werden können, beschreiben die folgenden Abschnitte.

7.2.1 Entwurf des Anwendungskerns I

Aus der Analysephase 7.1 erhält man eine Liste kontextabhängig dynamischer Anforderungen. Diese können zunächst durch herkömmliche Entwurfsprozesse als Komponenten realisiert werden. Dabei ist lediglich zu beachten,

dass mithilfe des Framework aus Kapitel 6 eine dynamische Anforderung später auf einen Aktuator vom Typ `ServiceProxy` (Dienst) des Framework im Teilsystem der Kontextadaption abgebildet wird. Dies erlaubt es, die daran gebundenen Komponenten zur Laufzeit hinzuzufügen, zu entfernen oder ihre Implementierung zu wechseln.

Interne oder externe Realisierung

Soll eine Komponente systemintern realisiert werden, muss darüber hinaus zu jedem `ServiceProxy` eine `Service` Komponente entworfen werden, an die der Dienst permanent gebunden wird. Bei extern zu realisierenden Komponenten wird von der Existenz einer solchen Komponente im Moment ihrer Verwendung bereits ausgegangen.

Die Entscheidung zwischen einer internen oder externen Realisierung ist in der Regel eine Frage der Ressourcen und der gewünschten Ubiquität. Bezogen auf die Hardware kann ein ubiquitäres System eine Kernhardware besitzen, welche die minimale (und situationsunabhängig verfügbare) Ausführungsplattform definiert. Dies ist oft der Fall, wenn es sich um ubiquitäre Anwendungen handelt, die nur in Zusammenhang mit einem bestimmten Gegenstand benutzt werden. In der Regel ist das ein bestimmtes mobiles Zugangsgerät wie ein Mobiltelefon. Es kann sich aber beispielsweise auch um eine standardisierte Minimalausstattung eines Autos handeln. Andere ubiquitäre Anwendungen haben dagegen keine Kernhardware oder sogar Kernsystem und können sich als mobiler Dienst auf eine beliebige Ausführungsplattform laden und dort realisieren.

Eine interne Realisierung eines Dienstes im Kernsystem ist angezeigt, wenn die Funktionalität in jeder Situation zur Verfügung stehen soll, oder die Realisierung ausschließlich auf die Kernhardware zurückgreift. Ein gutes Beispiel dafür ist das Framework selbst.

Eine externe Realisierung kommt dagegen in Frage, wenn die Funktion nicht in jeder Situation benötigt wird und ihre Realisierung Ressourcen benötigt, die nicht von der Kernhardware bereitgestellt werden können. Beispiele sind etwa die Anzeigefunktionen stationärer Bildschirme und Ähnliches.

Technischer Kontext

Wie bereits eingangs von 7.2 erwähnt, können bestimmte Formen der Realisierung zusätzliche Adaptionen und damit Abhängigkeiten von einem Kontext erzeugen. Im Fall des Kernsystementwurfes betrifft das sekundäre externe Abhängigkeiten von Kernsystemkomponenten, die beispielsweise unter

Adaption der Kosten und der Qualität zur Laufzeit an die Umgebung gebunden werden.

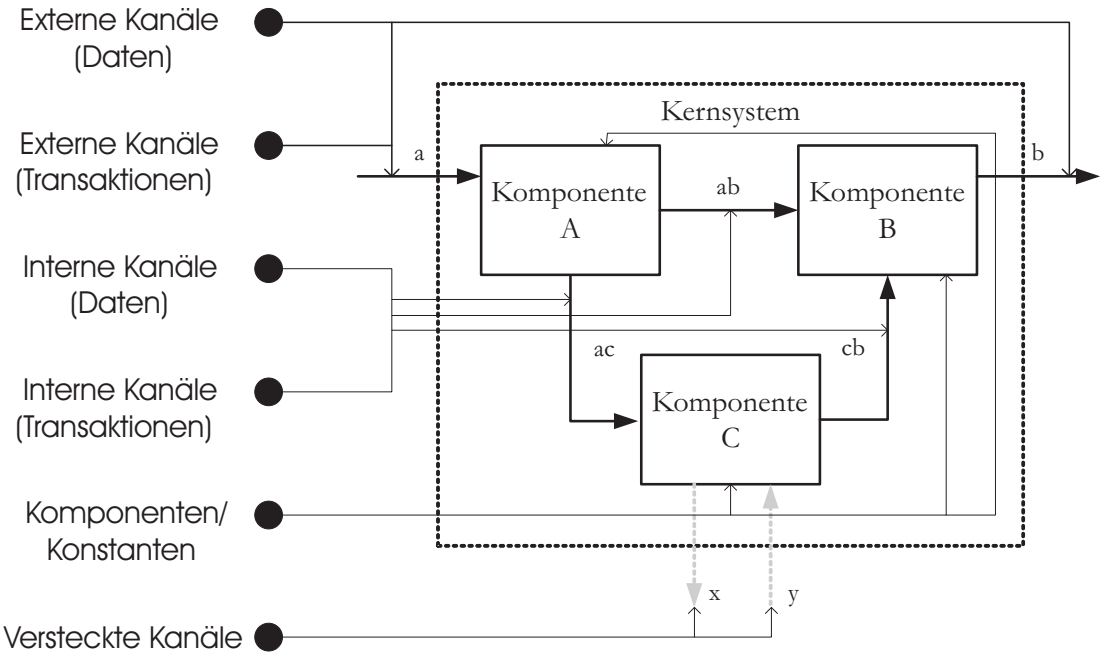


Abb. 42: Kandidaten für technische Kontextinformationen

Generell kommen als potenzielle Kandidaten für technischen Kontext aber prinzipiell alle technisch angebundenen Umgebungsinformationen in Frage. Das sind Informationen, die während eines gegebenen Zeitintervalls von außen in ein betrachtetes System gelangen, oder von diesem beeinflusst werden (Abb. 42):

- Die Summe der über die Ein- und Ausgabekanäle der einzelnen Komponenten transportierbaren Informationen. [Kanalbelegungen als mögliche Kontextinformationen].
- Alle Informationen, dass über einen Kanal eine Nachricht bestimmten Typs gesendet oder empfangen wurde.
- Die Information, aus welchen Komponenten das Kernsystem besteht, inklusive der Abhängigkeiten zwischen den Komponenten (folgt aus den ersten beiden Punkten).

Auch versteckte Kanäle können Kontext sein.

Dass auch die lediglich intern kommunizierten Informationen technische Umgebungsinformationen enthalten können, ist durch die Tatsache begründet, dass Komponentenspezifikationen manchmal versteckte Kanäle enthalten. Für versteckte Eingabekanäle äußert sich das etwa in spontanen, das heißt nicht aus der Modellierung erkennbaren Veränderungen des internen

Zustandes einer Komponente. Diese Form der versteckten Kommunikation ist oft bei Komponenten zu finden, die Zugriffe auf Hardware kapseln und deren Zustand beispielsweise gemeinsam mit Hardwarekomponenten genutzte Speicherbereiche (I/O Ports etc.) enthält.

Informationen über die innere Struktur des Kernsystems, hier also die Spezifikation der Teilkomponenten und Abhängigkeiten können ebenfalls Teil des technischen Kontextes sein. Dies entspricht der Definition von Kontext, da der Zustand des Kernsystems wie die Benutzerbedürfnisse auch Teil der umgebenden Situation sein kann, wenn dies für das System selbst relevant ist. Dies ist dann der Fall, wenn die Information dazu verwendet wird, die innere Struktur des Kernsystems zu adaptieren. Im Framework findet diese Adaption aber über separate Steuerkanäle statt, die vom Teilsystem der Kontextadaption in das Kernsystem führen. Dieser Fall ist also bereits in den ersten beiden Bedingungen enthalten.

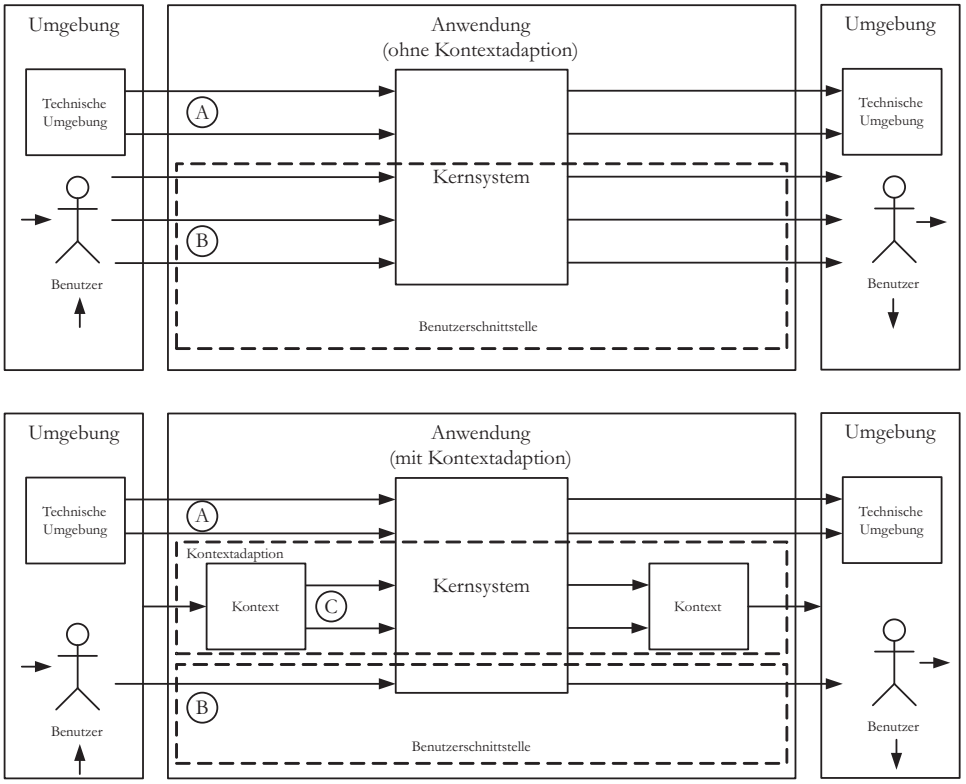


Abb. 43: Umgebung als Schnittstelle, Benutzerschnittstelle oder Kontext

Welche Umgebungsinformationen des Kernsystems letztendlich nun als technischer Kontext betrachtet werden, ist eine Entwurfsentscheidung, vergleichbar der Entscheidung in nichtubiquitären Systemen, welche Umgebungsinformationen über externe Komponenten (Abb. 43 A) oder die Benutzerschnittstelle (Abb. 43 B) eingebunden werden. Für den Entwurf des

Kernsystems ubiquitärer Anwendungen erweitert sich diese Entscheidung lediglich um eine weitere Klasse, die eine indirektere Art der Realisierung einer Interaktion zwischen System und Umgebung beschreibt (Abb. 43 C).

Bereits bei der Formalisierung 5.3 und eingangs des Kapitels wurde argumentiert, dass sich für diese Entwurfsentscheidung keine natürlichen formalen Kriterien definieren lassen, solange sowohl die Begriffe Kontext, Adaption als auch Umgebung relativ zu einer willkürlichen Systemgrenze definiert werden. Die folgenden Entwurfskriterien basieren daher lediglich auf der Zielsetzung der Ubiquität und diesbezüglichen Erfahrungswerten:

1.) Informationen über die Auswahl einer Komponente, die zur Laufzeit an einen im Kernsystem genutzten Dienst gebunden werden soll (also die Informationen auf dem Steuerkanal der von `ServiceProxy` abgeleiteten Komponenten), gehören (wie zuvor bereits diskutiert) zum Kontext, falls das Kernsystem wie die Kontextadaption aus Diensten konstruiert ist.

2.) Abhängigkeiten des Kernsystems von der Umgebung, die sich nicht als Dienst ausdrücken lassen, etwa weil kein standardisiertes syntaktisches Interface angegeben werden kann gehören zum Kontext. Zum Beispiel weil die Ein- oder Ausgabe einer Information von sehr unterschiedlichen Geräten stammen kann.

3.) Informationen, die normalerweise über eine Benutzerschnittstelle eingebunden würden, aber auch (aktuell) in Situationen zur Verfügung stehen sollen, in denen kein Benutzer existiert, oder keine explizite Interaktion mit dem Benutzer über eine direkte (bewusste) Benutzerschnittstelle möglich ist, sind Kontext.

4.) Informationen aus/für externe Komponenten, die möglicherweise nicht gleichzeitig mit der empfangenden/sendenden Komponente des Kernsystems existieren. Ein Beispiel dafür wären über drahtlose Netze verbundene externe Komponenten. Deren Verbindung könnte just im Moment ihrer Verwendung unterbrochen sein. Anstatt in jeder einzelnen Komponente einen Zwischenspeichermechanismus zu realisieren, kann man von der Entkoppelungseigenschaft des Kontextes gebrauch machen.

7.2.2 Entwurf der Kontextadaption

Mithilfe dieser Entwurfskriterien ergibt sich aus dem Entwurf des Kernsystems ein zusätzlicher Kontext (technischer Kontext) des Kernsystems, der das Ergebnis von weiteren Adaptionsentscheidungen ist, die zu den Adaptio-

nen anhand der Gültigkeitsbedingungen dynamischer Anforderungen (also dem Nutzerkontext aus der Kontextanalyse) hinzukommen.

Kontext und Adaptionen finden sich später vereint im Teilsystem der Kontextadaption. Das Framework aus Kapitel 6 hält dafür bereits eine generische Realisierung eines kalibrierbaren Modells für Kontextadaption (K-Modell) bereit. Für den Entwurf der Kontextadaption einer konkreten ubiquitären Anwendung muss also nur noch eine geeignete Initialisierung des Modells mit einer Anfangsbelegung entworfen werden.

Dies geschieht in Form einer K-Modell Beschreibung nach 6.3 als XML Dokument. Zur Veranschaulichung kann daher auch auf die grafische Beschreibungstechnik aus 6.4 zurückgegriffen werden. Die K-Modell Beschreibung dient dann als Initialkontext der konkreten ubiquitären Anwendung.

Definition 22: *Initialkontext*

Der Initialkontext ist die anfängliche Belegung des Kontextzustandes eines betrachteten Systems.

Der Initialkontext initialisiert auch die Kontextadaption.

Der Initialkontext ist deshalb von so großer Bedeutung, da die K-Modell Realisierung des Framework als ihr eigenes Metamodell fungiert, der Initialkontext also auch die Beschreibung eines Modells der Kontextadaption mit einschließt. Die Forderung nach Kalibrierbarkeit machte es nämlich notwendig, dass ein K-Modell seine eigene Spezifikation als Kontext enthält und selbst (über den Aktivator) realisieren kann. Der dem Initialkontext entsprechende Anfangszustand des Kontextes spezifiziert daher auch das gesamte weitere Adaptionsverhalten.

Der Initialkontext enthält beispielsweise eine Liste der verwendeten Sensorendienste sowie jeweils einen Identifikator auf eine instanzierte Komponente (Agent), welche den Dienst erbringen kann. Beim Start der Anwendung holt sich der Aktivator (der ein Aktuator der Bootadaption ist, die eine vollständige Kalibrierung erlaubt) diese Daten aus dem Kontext, aktiviert die dort angegebenen Sensordienste im Teilsystem der Kontextadaption und bindet diese an die entsprechenden externen oder internen Komponenten. Dabei kann auch die Bootadaption mit anderen Kalibrierungsadaptionen überschrieben werden, welche für die konkrete Anwendung geeignet sind (siehe auch 6.1.5).

Gerätebezogen könnte man sich den Vorgang also auch als einen generischen Bootloader vorstellen, der beim ersten Start des Gerätes nach einer Firmware sucht, diese herunterlädt und sich damit selbst überschreibt.

Der Entwurf solcher initialen K-Modellbeschreibungen beginnt zunächst mit den Gültigkeitsbedingungen der dynamischen Anforderungen aus der Kontextanalyse (7.1) und dem technischen Kontext aus dem Entwurf des Kernsystems 7.2.1. In beiden Fällen kann die Kontextinformation als Ergebnis einer Adaption betrachtet werden, die nur noch in die Realität des Systems umgesetzt werden muss. Nach den Strukturierungsvorgaben aus 6.4.3 heißen solche Kontextinformationen Adaptionskontext.

Entwurf des Adaptionskontextes und seiner Aktuatoren

Der Adaptionskontext definiert indirekt die Kommunikationsschnittstelle zwischen den beiden Teilsystemen der Kontextadaption und des Kernsystems. Der Adaptionskontext ist das Modell bestimmter Aspekte der Umgebungssituation, die einen Soll-Zustand respektive eine Zielvorgabe darstellen, welche durch die nächste Adaption erreicht werden soll. Dabei kann es sich um Änderungen handeln, die aus erkannten Benutzerbedürfnissen resultieren (Nutzbarkeitsdimension der Ubiquität) oder welche die Verfügbarkeit einer gerade benötigten Funktionalität oder Qualität derselben sicherstellen soll (Verfügbarkeitsdimension/technischer Kontext). Der Adaptionskontext lässt sich bei der Modellierung gemäß 5.1.2 in mehrere Elemente gruppieren, die sich formal als Dienste interpretieren lassen. Das Framework hält dann mit dem Kontextserver eine Komponentenimplementierung bereit, an die gleich mehrere dieser Dienste gebunden werden können. Über diese generische Datenaustauschkomponente wird im Falle des Adaptionskontextes die Schnittstelle zwischen den beiden Teilsystemen Kontextadaption und Kernsystem abgewickelt. Umgesetzt wird die Schnittstelle durch die Aktuatoren. Dabei handelt es sich wie in Def. 17 (5.3.5) festgelegt um Dienste, die den Zugriff auf einen bestimmten Teil des Adaptionskontextes erlauben. Die Verbindung zum Kernsystem wird hergestellt, indem eine Komponente aus dem Kernsystem zur Laufzeit oder permanent an den Dienst (realisiert durch einen Zugriffsproxy im Teilsystem der Kontextadaption) gebunden wird. Für den Entwurf der Aktuatoren gibt es folglich verschiedene Möglichkeiten:

- Es wird ein neuer Aktuator (d.h. Dienstproxy) konstruiert, der an die entsprechende Komponente des Kernsystems gebunden werden kann, welche den Adaptionskontext direkt konsumieren soll. Dazu muss allerdings die zu bindende Komponente des Kernsystems modifiziert werden, so dass sie das `Actuator` Interface des Framework erfüllt. Ist das nicht möglich, weil es sich etwa um eine fertige Komponente handelt, muss ein entsprechender Wrapper konstruiert werden, der das Aktuatorinterface bereitstellt.

- Alternativ kann ein generischer Aktuator des Framework verwendet werden. Das Framework hält drei solcher Aktuatoren bereit (siehe 6.2). Der erste realisiert einen Zugriff (poll) auf beliebige Kontextinformationen. Die anderen beide funktionieren ähnlich dem Aktivator, der die Adaption der Kontextadaption für die Kalibrierung realisiert, wirken allerdings im Unterschied dazu auf das Teilsystem des Anwendungskerns. Der eine Aktuator erlaubt das Senden beliebiger Nachrichten an beliebige Komponenten des Kernsystems. Im zugehörigen Adaptionskontext muss lediglich der Identifikator der Komponente, der Name der aufzurufenden Methode sowie die einzelnen Parameter vermerkt sein. Der zweite Aktuator arbeitet exakt wie der Aktivator, allerdings ist das Hinzufügen und Entfernen von Diensten respektive deren Bindung nicht auf Sensoren, Interpreter, Aktuatoren und Kontextelemente beschränkt, sondern funktioniert für beliebige Komponenten des Anwendungskerns.

Aufgrund der daraus resultierenden Komplexität bei Einsatz eines Kernsystem Activators (im Unterschied zum Kontextadaptions-Activator, der nur auf das Teilsystem der Kontextadaption und seine vier Diensttypen beschränkt ist) werden diese generell wie auch in der Fallstudie nur in Spezialfällen eingesetzt. Meistens, wenn es darum geht, auch das Framework selbst zur Laufzeit einer darauf basierenden Anwendung von einer HW-Plattform auf eine andere (oder sogar von Framework zu Framework) migrieren zu können. Ansonsten verzichten ubiquitäre Anwendungen auf eine totale Rekonfigurierbarkeit des Gesamtsystems und begnügen sich mit der vollständigen Rekonfigurierbarkeit des Teilsystems der Adaption für die Kalibrierung. Das Kernsystem besteht dagegen oft aus einer statischen Dienstarchitektur, bei der zwar die Implementierungen gewechselt, neue Dienste aber nur indirekt über die Dienstabhängigkeiten gebundener Komponentenimplementierungen (und damit in deren Verantwortungsbereich) hinzugefügt oder entfernt werden können (partiell rekonfigurierbares Teilsystem).

Adaptionsaktionen

Adaptionsaktionen sind das Äquivalent wiederverwendbarer Komponenten für Kontexte. Sie finden ihrer Anwendung bei der effizienten Modellierung häufig wiederkehrenden Berechnungen im Adaptionskontext besonders von generischen Aktuatoren. Der Grund dafür ist, dass generische Aktuatoren in der Regel eine technisch abstrakte aber detaillierte Beschreibung des Sollzu-

standes verarbeiten, beispielsweise in Form einer Spezifikation der zu aktivierenden Komponenten und ihrer Kommunikationsverbindungen. Diese Spezifikationen müssen im Adaptionskontext hinterlegt werden und tauchen dort sehr oft mit nur geringen Unterschieden wiederholt auf. Durch Darstellung als parametrisierte Templates in Form eines Interpreters sowie eines Template und eines Parameterkontextes. Solche Modellierungsmuster des Adaptionskontextes werden Adaptionsaktionen genannt und können der besseren Übersichtlichkeit halber als syntaktisches Makro (siehe auch 6.4.3) definiert werden, das die Kontextelemente der Transaktion und der zu übertragenen Parameter gruppiert.

Neben technischen Templates für generische Aktuatoren kommen Adaptionsaktionen auch immer dann zum Einsatz, wenn die durchzuführende Änderung technisch relativ unspezifisch ist und ihre technische Realisierung teilweise im Kontext berechnet werden soll. Dies ist beispielsweise der Fall bei den als Kontext ausgedrückten Anforderungen, deren Umsetzung im Anwendungskern weiteren technischen Kontext erzeugt hat. Adaptionsaktionen gruppieren solche im Kontext abgebildeten Berechnungen der technischen Realisierung wiederum aus Übersichtlichkeitsgründen. Sie repräsentieren aber auch einen Entwurfsschritt, der aus eben der Definition einer solchen technischen Realisierung im Kontext besteht (Abb. 44 veranschaulicht das Prinzip).

Strukturierungsmerkmal "Aktion".

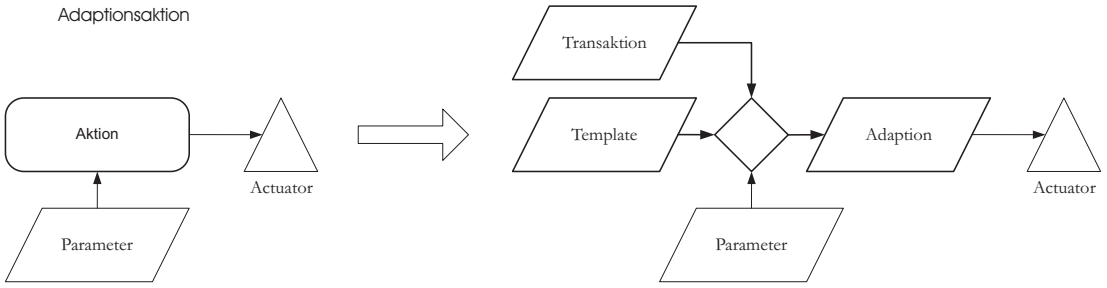


Abb. 44: Prinzip der Adaptionsaktion

Eine Adaptionsaktion besteht dabei aus einem Kontextelement, das die Transaktion ihrer Aktivierung repräsentiert und ein oder mehreren optionalen Parametern und Interpretern zur Berechnung eines Adaptionskontextes.

Für jeden Interpreter muss wie zuvor bei den Aktuatoren ein Dienstproxy entworfen werden, der dann zur Laufzeit an eine externe oder interne Realisierung gebunden werden kann.

Situationskontext

Neben der Festlegung von Aktionen, die zur Adaption ausgeführt werden, können auch Situationen im Modell der Kontextadaption abgebildet werden. Genau genommen ist zwar der gesamte Kontext ein Modell einer Situation nach Definition 8 (3.2.3). Zum Zwecke einer intuitiven Beschreibung kann der Begriff der Situation aber auch zur Strukturierung des Kontextes verwendet werden. Der Situationskontext trennt dabei das Modell der äußeren Situation im Kontext von den daraus abgeleiteten Adaptionsaktionen.

Strukturierungsmerkmal "Adaptor".

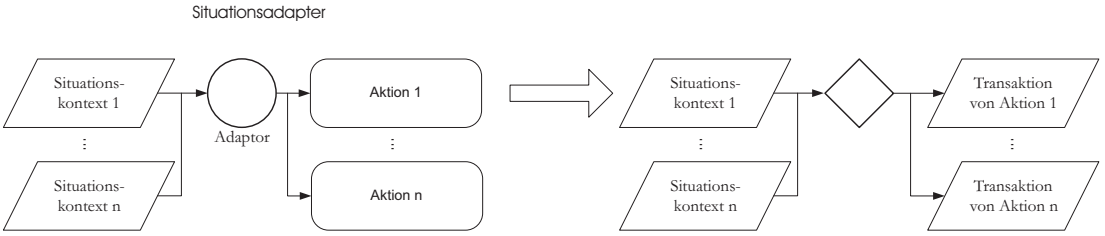


Abb. 45: Situationsadaptoren

Verknüpft werden können die beiden Bereiche des Kontextes durch Adaptoren. Dabei handelt es sich wie im Fall von Adaptionsaktionen lediglich um ein Beschreibungsmittel (Abb. 45). Ein Adaptor ist ein Interpreter, der eine Situation repräsentiert, die aus dem Situationskontext erkannt werden kann und die mit bestimmten Adaptionsaktionen verknüpft werden soll.

Der Entwurf des Situationskontextes richtet sich dabei nach der Auswahl des an den Interpreter gebundenen Entscheidungsalgorithmus. In der Regel wird dazu eine existierende Komponente verwendet, beispielsweise ein Regelinterpreter, ein Neuronales Netzwerk oder ein Entscheidungsbaum. Das Framework hält einige häufig und generisch verwendbare Realisierungen für Interpreter bereit, zum Beispiel einen einfachen Regelinterpreter. In solchen Fällen richtet sich die Art des Situationskontexts auch daran aus, was für den verwendeten Algorithmus günstig ist. Der Situationskontext muss aber keinesfalls direkt durch verfügbare Sensoren messbar sein. Zu beachten ist lediglich, dass aus Gründen der leichteren Nachvollziehbarkeit, eine nahe liegende Verbindung zwischen Situationskontext und erkannter Situation existieren sollte. Auf diese Weise wird verhindert, dass die Modellierung einer Situationserkennung zu viele versteckte Annahmen enthält (siehe auch Balancierungsmetriken in 7.1.5).

Das Thema der Modellierung impliziter Annahmen und ihrer Kalibrierung wird in einem der nächsten Abschnitte (7.3) noch einmal genauer behandelt.

Von geöffneten Fenstern eines Raumes beispielsweise auf eine Situation zu schließen, in welcher der Benutzer keine Sprachsteuerung, sondern lieber einen Bildschirm verwenden will, ist unter bestimmten Annahmen möglich. Möglicherweise befindet sich die einzige mögliche Lärmquelle außerhalb. Ohne genaue Kenntnis der impliziten Annahmen ist es aber schwer, diese Modellierung nachzuvollziehen.

Ist dagegen die Interpretationskette durch mehrere Zwischenschritte und dazwischenliegende Interpreter modelliert, werden die zugrunde liegenden Annahmen leichter sichtbar. Ein besserer Situationskontext in diesem Fall besteht also in der Information, ob sich der Benutzer in einer lauten Umgebung befindet. Die Annahme, dass die Anwesenheit in einem bestimmten Raum bei geöffnetem Fenster zu einer lauten Umgebung führt, kann dann über eine explizite Kontextberechnung ausgedrückt (und später leicht geändert, d.h. kalibriert) werden.

Sensorkontext

Sofern es sich nicht um über den Initialkontext zum Zeitpunkt des Entwurfes festgelegte Konstanten handelt, müssen alle Kontextelemente, deren Informationen nicht aus anderen berechnet werden mit einem Sensor versehen werden (Abb. 46).

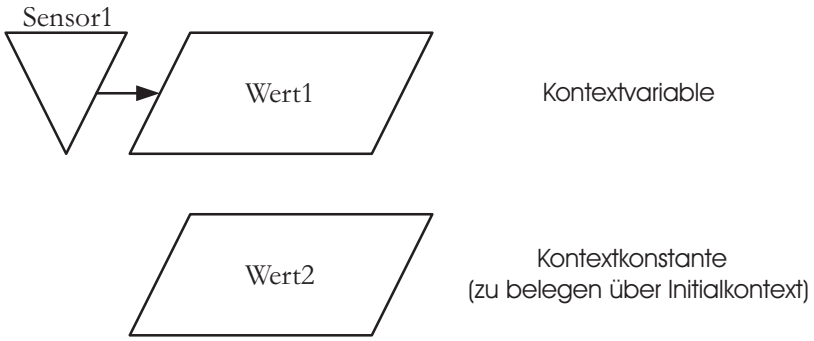


Abb. 46: Sensorkontext vs. Kontextkonstante

Solche Kontextelemente gehören zum Sensorkontext. Sensoren sind wieder formal als Dienste zu sehen, die zur Laufzeit an eine Komponente innerhalb oder außerhalb des betrachteten Kernsystems gebunden werden können. Im Framework kommunizieren diese Sensorkomponenten entkoppelt über die diensterfüllenden Komponenten der verbundenen Kontextelemente, in der Regel also über den Kontextserver.

Manchmal können die für den Situationskontext oder im Adaptionskontext als Parameter benötigten Informationen aber nicht direkt durch einen Sensor erzeugt werden. Entweder weil eine geeignete Komponente gerade für das betrachtete System nicht zur Verfügung steht, oder weil die gewünschte Information aus technischen Gründen prinzipiell nicht messbar ist. Der Begriff *messbar* ist dabei aber sehr weitreichend zu interpretieren. Die explizite Eingabe der Bedürfnisse eines Benutzers durch direktes Nachfragen ist ebenso ein gültiger Sensor wie die physikalische Messung einer Temperatur. Existiert aber kein geeigneter Sensor, kann die benötigte Information durch ein oder mehrere Berechnungsschritte in Form von Interpretern aus einem anderen Sensorkontext abgeleitet werden. Die Interpreter beschreiben dabei wieder interne oder externe Komponenten, die zur Laufzeit an den jeweiligen im K-Model beschriebenen Dienst (Interpreter) gebunden werden können. Dabei kann es sich um spezialisierte Interpreter (z.B. °F->°C) oder auch um allgemeine Komponenten handeln, die beispielsweise auf Basis von Ontologien und logischer Deduktion eine Kontextinformation in eine andere überführen.

Kontext kann vor seiner Verwendung interpretiert werden.

Konsistenz

Eine wichtige Eigenschaft im Zusammenhang mit dem Entwurf von Kontextinterpretationen, beziehungsweise ganz allgemein der Kontextadaption ist die Konsistenz der angegebenen Spezifikationen. Neben der syntaktischen Konsistenz bedeutet dies auch, dass die Spezifikation keine widersprüchlichen bzw. unerfüllbaren Annahmen und Bedingungen enthalten darf, da für sie ansonsten keine Lösung in Form einer funktionierenden Implementierung existiert (semantische Konsistenz).

Die Überprüfung der Konsistenz zu einem bestimmten Zeitpunkt (d.h. für einen konkreten Kontext) ist prinzipiell möglich. Die Spezifikation eines Kontextadaptionsmodells lässt sich ja wie in 5.3 definiert in ein gängiges formales Basismodell übersetzen und mit den dafür existierenden Beweiswerkzeugen auf die Gültigkeit bestimmter Aussagen, wie der Konsistenz (d.h. der Existenz einer Lösung) hin überprüfen. Der Aufwand für die Transformation in die für die maschinelle Bearbeitung notwendigen Normalformen und die Interpretation der Ergebnisse sind aber erheblich, besonders im Hinblick darauf, dass ein Resultat immer nur für einen bestimmten Kontext gilt und der gesamte Vorgang bei jeder Kalibrierung erneut und vollautomatisch durchgeführt werden müsste.

Als Alternative zu einer aufwändigen gesamtheitlichen Konsistenzprüfung (syntaktisch, semantisch etc.) lassen sich aber einige der für das Modell von

Kontextadaption geltenden Anforderungen und einige konstruktive Merkmale für eine starke Vereinfachung der Konsistenzproblematik ausnutzen:

- Das Modell ist gemäß den Ubiquitätsbedingungen unempfindlich gegenüber lokalen Inkonsistenzen.
  Dies liegt darin begründet, dass der Kontext als optional (Verfügbarkeitsdimension) für die Ausführung des Kernsystems definiert wurde. Das Fehlen von Kontextinformationen beispielsweise durch den Defekt eines Sensors darf zwar das Verhalten des Kernsystems beeinflussen, nicht jedoch in einen Fehlerzustand führen, das heißt schlimmstenfalls (alle Spezifikationen von allen Elementen des Kontextes sind nicht in Form einer Implementierung lösbar) wartet das Kernsystem auf darauf, dass sich dieser wieder Zustand ändert. Für kontextabhängige Systeme ist aber erst einmal nicht unterscheidbar, ob der Umstand, dass für eine lokale Spezifikation eines Elementes wie einem Sensor keine Implementierung gefunden werden kann, an der Spezifikation generell oder nur für eine bestimmte Situation (d.h. gerade keine Implementierung vorhanden) inkonsistent ist.
- Die einzelnen Elemente des Kontextmodells sind voneinander entkoppelt. Ist die Spezifikation eines einzelnen Elementes inkonsistent und hat deshalb keine Lösung, beeinflusst dies die anderen Elemente höchstens indirekt, führt aber nicht zu einer Inkonsistenz des gesamten Modells, da jede lokale Spezifikation eines Elementes sich höchstens auf den verbundenen Kontext, nicht jedoch auf andere Interpreter, Sensoren oder Aktuatoren beziehen darf.
- Der Zustand eines Kontextelementes ist immer konsistent, da lediglich eine Quelle (Sensor oder Interpreter) pro Kontextelement zugelassen ist. Dadurch könnten Inkonsistenzen nur lokal in der Quelle selbst (und ihrer Spezifikation) verursacht werden. Nach den Anforderungen ist die Modellierung aber unempfindlich gegenüber lokalen Inkonsistenzen.

Die Überprüfung der einfachen Konsistenz eines Modells für Kontextadaption reduziert sich mit diesen Festlegungen auf eine Überprüfung der lokalen Konsistenz der technischen Detailspezifikationen einzelner Elemente (`sy_type`) sowie einer Überprüfung des syntaktischen und semantischen Typs von Kontextinformationen zwischen einer Zugriffsrolle (Sensor, Interpreter, Aktuator) und den in einem Kontextelement gespeicherten Daten.

Beide Bedingungen lassen sich lediglich auf Basis der technischen Detailspezifikation eines Elementes entscheiden oder semi-entscheiden (eine Elementspezifikation ist lokal konsistent, wenn sich eine Implementierung finden

lässt, welche die Spezifikation erfüllt) und können daher nur in den Modellaktuatoren oder im Fall der Typprüfung für Kontextinformationen auch in den Implementierungen der Elemente selbst überprüft werden. Nur auf diese Weise lässt sich die Anforderung nach einer Unempfindlichkeit gegenüber lokalen Inkonsistenzen realisieren.

Indirekt ist auch die globale semantische Konsistenz über Tolerierbarkeit lokaler Inkonsistenzen sichergestellt. In manchen Fällen reicht diese einfache Konsistenzzusicherung jedoch nicht aus, um gewünschte Eigenschaften eines Systems zu spezifizieren. In vielen Formen technischer Spezifikation können zu diesem Zweck neben den allgemeinen auch fallweise spezifische Konsistenzbedingungen definiert werden, die mehrere Elemente miteinander verknüpfen, beispielweise durch die Definition zusätzlicher Axiome in OCL Constraints der UML. Wie bereits angedeutet, sind solche Elemente übergreifenden Konsistenzbedingungen innerhalb des Modells von Kontextadaption aber nicht zulässig und auch nicht möglich, da sich die jeweils pro Element des Kontextmodells verwendeten technischen Detailspezifikationen (also die hinter `sy_type` verbergenden Dokumente) voneinander unterscheiden können und die Integrationsfunktion (d.h. der Modellaktuator) jeweils nur in eine Richtung und beschränkt auf die für Kontextadaption wichtigen Teile des Systemmodells definiert ist. Selbst im in der Praxis wahrscheinlicheren Fall der Verwendung einer einheitlichen Form der technischen Detailspezifikation würde durch die Formulierung von beliebigen Konsistenzbedingungen zwischen Elementen des Kontextadaptionsmodells auf der technischen Ebene die Entkoppelungsbedingung verletzt. Lokale Inkonsistenzen könnten sich dann beliebig fortpflanzen bzw. es könnten neue Inkonsistenzen durch die Kombination von Elementen mit spezifischen Konsistenzbedingungen entstehen.

Als Ausweg können globalen Konsistenzbedingungen und mögliche Reaktionen auf ihre Verletzung als explizite Kontextadaption modelliert werden. Auf diese Weise kann auch die Art des Umgangs mit lokal auftretenden Inkonsistenzen genauer spezifiziert werden. Diese erweiterten Konsistenzbedingungen werden in Form von Interpretern modelliert. Die Ergebnisse stehen dann wieder im Kontext zur Verfügung und können weiterverarbeitet werden. Das Prinzip wird im Folgenden anhand einiger wichtiger Beispiele erklärt.

Redundante Sensoren

In manchen Fällen kann es notwendig sein, mehr als einen Sensor für einen einzigen Sensorkontext vorzusehen, etwa weil die Messwerte eines Sensors fehlerhaft sein, oder ein Sensor ausfallen könnte. Die Spezifikationstechnik

aus 6.3 erlaubt aber für die Modellierung aus Konsistenzgründen lediglich eine Quelle pro Kontextelement. Der Grund dafür liegt genau in der gerade beschriebenen Fehleranfälligkeit realer Sensoren. Es könnte der Fall eintreten, dass zwei Sensoren unterschiedliche Messwerte für eine Kontextinformation liefern. Wären diese mit dem gleichen Kontextelement verbunden, würde dessen Zustand ständig zwischen den beiden Messwerten pendeln und eine Veränderung der Umgebungsinformation signalisieren, die nicht in dieser Form stattfindet. In Wahrheit beschreibt ein Kontextelement zunächst aber nur den Messwert eines Sensors und nicht eine absolute Wahrheit über die Umgebungssituation. Diese wird vielmehr aus den Kontextinformationen über die Messwerte interpretiert.

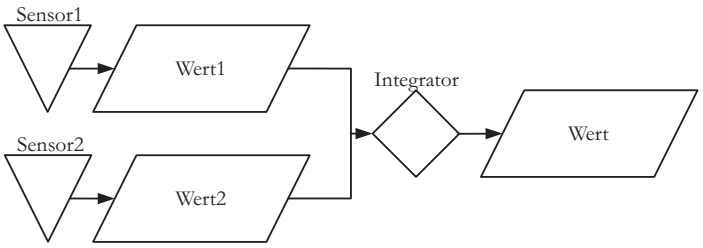


Abb. 47: Entwurf redundanter Sensoren

Abb. 47 zeigt demgemäß die richtige Anbindung mehrerer Sensoren für ein und dieselbe interpretierte Umgebungsinformation. Die Werte redundanter Sensoren werden dabei zunächst als eigene Kontextelemente modelliert. Ein separater Interpreter schließt dann aus den Zuständen der beiden Elemente auf den Zustand der realen Situation, beispielsweise mithilfe der Bildung des Mittelwertes oder eines vergleichbaren Verfahrens. Geeignete Realisierungen für Integrationsinterpreter müssen zur Laufzeit wiederum an den Interpreterdienst gebunden werden. Das Framework enthält bereits Realisierungen für einfache Integrationen redundanter Sensoren (Mittelwert und Fallback).

Auflösung von Adaptionskonflikten

Die analoge Umkehrung des Problems der redundanten Sensoren ist die Auflösung von Adaptionskonflikten. Wie der Sensorkontext, so ist auch der Adaptionskontext vor widersprüchlichen Belegungen dadurch geschützt, dass für ein Kontextelement lediglich eine einzige Quellverbindung zulässig ist. Können mehrere Verarbeitungspfade Ergebnisse für ein identisches Kontextelement im Adaptionskontext produzieren, müssen diese, vergleichbar

der Zusammenführung redundanter Sensormesswerte, erst durch einen Interpreter miteinander integriert werden (Abb. 49).

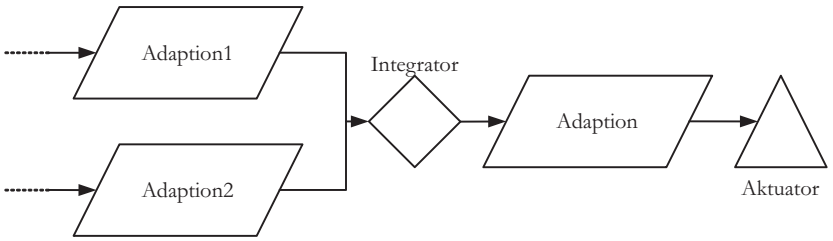


Abb. 49: Prinzip der Auflösung von Adaptionskonflikten

Nebenläufigkeit und Synchronisation

Im Vergleich zu anderen Beschreibungstechniken enthält die Beschreibung von Adaptionsprozessen aus 6.3 keine expliziten Grundelemente zur Erzeugung oder Synchronisierung von Nebenläufigkeit (vgl. Split/Join in UML Aktivitätsdiagrammen), weil die Elemente eines K-Modells bereits per Definition als voneinander entkoppelte aktive, d.h. selbstständig agierende und nebenläufige Komponenten festgelegt worden waren. Durch Erzeugung einer Kontextinformation in Form von Daten, die in einem Kontextelement abgelegt werden, können so durch einen Sensor oder Interpreter je nach Struktur des K-Modells voneinander abhängige oder unabhängige Ereignisketten (Datenflüsse) angestoßen werden. Diese Eigenschaft von K-Modellen kann man sich auch bewusst für die Steuerung (Einschränkung) der Nebenläufigkeit zunutze machen, indem man über spezielle Interpreter die nebenläufigen Datenflüsse in einem gemeinsamen Knoten (Kontextelement) aufspaltet oder vereinigt. Die Behandlung redundantet Sensorinformationen (Abb. 47) und von Adaptionskonflikten (Abb. 49) sind jeweils Spezialfälle einer solchen allgemeinen Synchronisation.

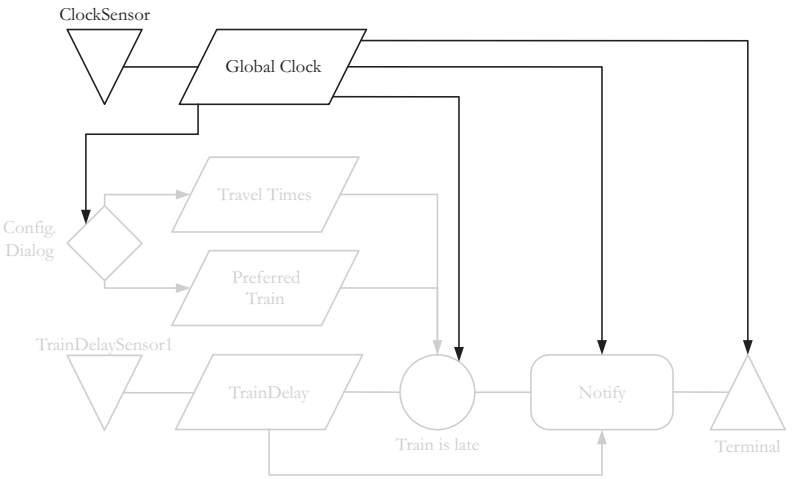


Abb. 48: Kontextadaption mit Globaler Uhr/Taktung

Zudem ist es möglich, für die Unterstützung von zeitbasierten Synchronisationsmaßnahmen explizit modellierte globale oder lokale Takte in Form von Zeitsignalsensoren und Kontexten hinzuzufügen (siehe Abb.48 und Abb.50).

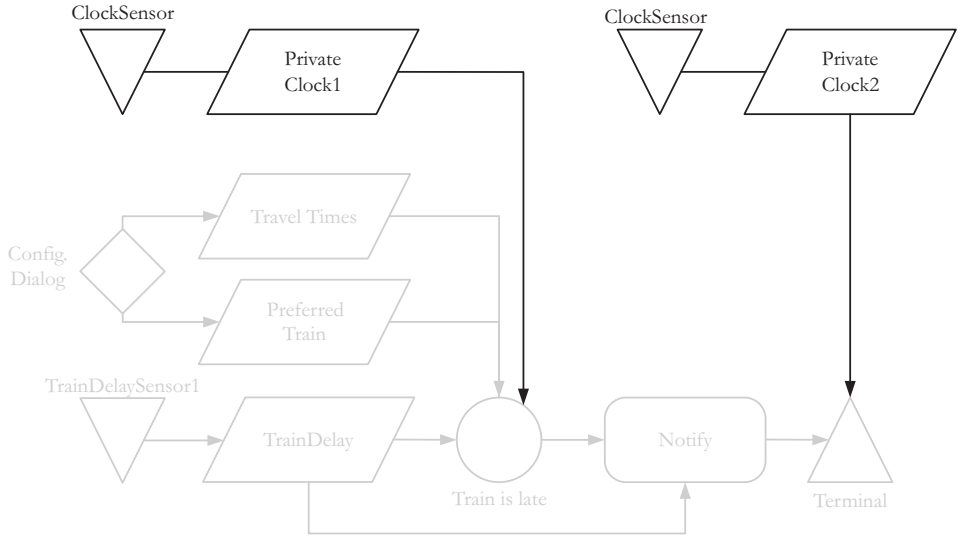


Abb. 50: Lokale Uhr/Taktung einer Kontextadaption

Bewachte Aktionen

Aufgrund der gerade erwähnten prinzipiellen Nebenläufigkeit der Informationsverarbeitung bei der Kontextadaption kann es notwendig sein, übergeordnete *Constraints* zwischen einzelnen Adaptionsaktionen zu modellieren. Damit kann verhindert werden, dass zwei sich gegenseitig ausschließende oder auf andere Weise interagierende, in Form von Adaptionsaktionen ausgedrückte Features zur gleichen Zeit ausgeführt werden. Zum Beispiel die Aktivierung einer Lampe bei gleichzeitiger Unterbrechung der Stromzufuhr.

Normalerweise wird die Ausführung von Aktionen lokal durch die jeweiligen Situationen bewacht, welche die betroffene Aktion auslösen können. Die als Situationen markierten Interpreter können die Informationen über die Auslösung potenziell konfliktträchtiger anderer Aktionen als Kontext in den eigenen Entscheidungsprozess mit einbeziehen (Abb. 51)

In manchen Fällen kann es aber notwendig sein, global gültige Wächter für Aktionen zu modellieren. Die Behandlung von Adaptionskonflikten durch einen vorgeschalteten Interpreter wie zuvor beschrieben ist so ein globaler Wächter. Andere Beispiele sind Transaktionsmonitore, Zusicherungen von

Sicherheitseigenschaften oder die Priorisierung von sich gegenseitig ausschließenden Aktionen (Abb. 52).

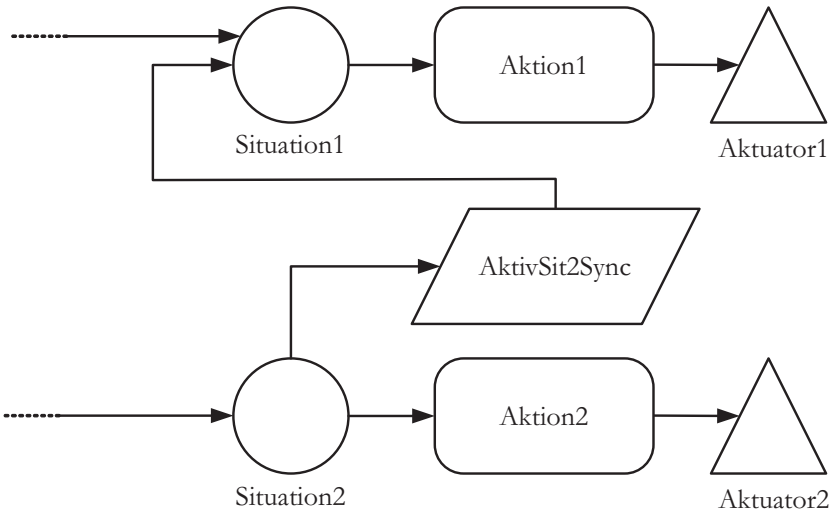


Abb. 51: Bewachte Situationen

7.2.3 Entwurf des Anwendungskerns II

Nach der Fertigstellung des Entwurfes der Kontextadaption muss in der Regel die Phase des Anwendungskernentwurfes wiederholt werden. Dabei werden die Realisierungen für die im letzten Schritt hinzugefügten Adaptionen festgelegt.

Zunächst muss wieder zwischen einer internen oder externen Realisierung unterschieden werden. Dabei ist aber darauf zu achten, dass die durch den technischen Kontext der ersten Entwurfsphase des Kerns bedingten Adaptionen vollständig intern realisiert werden. Externe Abhängigkeiten hätten sonst nämlich weiteren technischen Kontext und damit Adaptionen zur Folge. Außerdem könnten auf diese Weise leicht so genannte *Single Point of Failure* entstehen. Das bedeutet, dass wenn eine technische Adaption aufgrund fehlender externer Ressourcen gerade nicht verfügbar ist, automatisch auch alle davon abhängigen Anforderungsadaptionen betroffen wären, auch wenn für deren Realisierung die externen Ressourcen bereit stünden.

Iterative Auflösung zirkulärer Abhängigkeiten.

Die verfeinerten Anforderungs- und Kalibrierungsadaptionen können dagegen beliebig extern oder intern realisiert werden. Eine vollständig externe Realisierung erzeugt in diesen Fällen keinen weiteren technischen Kontext. Dieser ist nämlich bereits in der Kalibrierung enthalten. Anders liegt der Fall, wenn die Dienste für Sensoren, Interpreter etc. an Komponenten aus dem Anwendungskern gebunden, also intern realisiert werden, diese Realisierung aber wieder sekundäre externe Dienstabhängigkeiten besitzt. Unter solchen Umständen müssten die letzten beiden Entwurfsschritte (Hinzufügen tech-

nischer Adaption, Entwurf der Realisierung) iterativ wiederholt werden, bis alle diese zirkulären Abhängigkeiten aufgelöst sind.

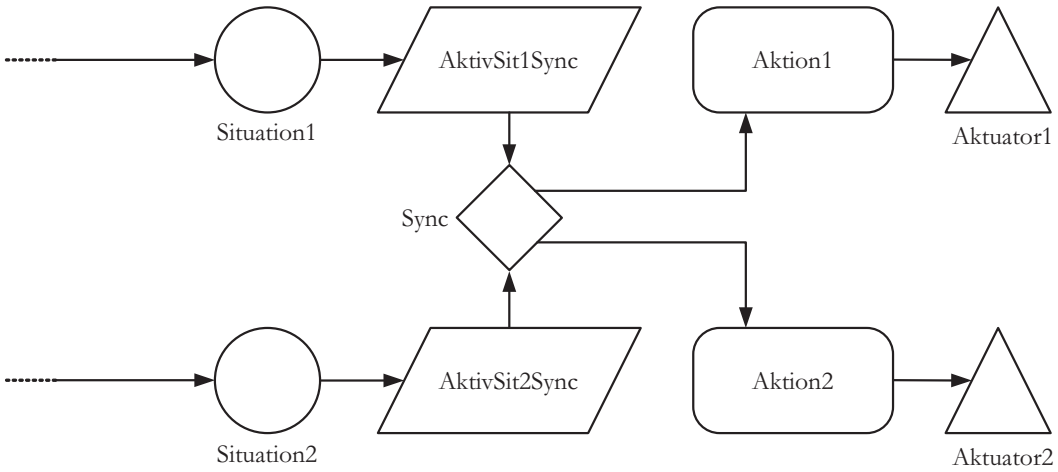


Abb. 52: Bewachte Aktionen

Ist der Entwurf der Realisierung des Kernsystems abgeschlossen, muss es nur noch mit der Realisierung des K-Modells des Framework kombiniert werden. Die in 7.2.2 entworfene Beschreibung des Teilsystems der Kontextadaption wird darin als Initialkontext gespeichert und wird dann von der Implementierung (durch den `Activator`=Modellaktuator) zur Laufzeit durch Rekonfiguration der K-Modell Realisierung des Framework automatisch realisiert, indem die Dienstproxies erzeugt und an externe Komponenten oder Komponenten des Kernsystems gebunden werden.

7.3 Entwurf der Kalibrierung

Kalibrierungen sind nach 5.1.1 lediglich spezielle Adaptionen, die sich von den normalen Adaptionen nur dadurch unterscheiden, dass sie das Teilsystem der Kontextadaption als Ziel haben, also die Ansammlung von Serviceproxies aller Kontextelemente, Sensoren, Interpreter und Aktuatoren. Die notwendigen Aktuatoren zur Modifizierung des Kontextadaptions-Teilsystems (`Activator` etc. siehe Kapitel 6) und deren Realisierung stellt das Framework bereits zur Verfügung.

Kalibrierung wird wie eine zweite Adaption entworfen.

Konstruiert wird die Kalibrierung wie die normale Kontextadaption eines ubiquitären Systems analog zu den in 7.1-7.2 beschriebenen Entwurfsschritten. Unterschiede ergeben sich lediglich in der Kontextanalyse, die sich auf die impliziten Annahmen konzentriert, die in dem zu kalibrierenden und bereits gegebenen Adaptionsmodell existieren. Diese werden zunächst explizit

modelliert, damit sie von der Kalibrierung leichter erfasst werden können (siehe 7.3.1). Unterschiede ergeben sich auch in der Modellierung der Umgebungsschnittstelle, da Kalibrierungen im Unterschied zu normalen Adaptionen häufiger auf direkte Interaktionen mit dem Benutzer zurückgreifen können und müssen, ohne die Ubiquität der eigentlichen adaptiven Anwendung zu gefährden, weil ja nur das stark entkoppelte Teilsystem der Kontextadaption Ziel der Kalibrierungsadaptionen ist (siehe auch 7.3.2).

7.3.1 Entwurf der Kalibrierungsschnittstelle (Kalibrierungskontext)

Wie in 7.2.2 definiert der Adaptionskontext die Schnittstelle zwischen Adaption und zu adaptierendem Teilsystem. Im Falle der Kalibrierung (Adaption der Adaption) sind Adaption und zu adaptierendes Teilsystem identisch (totale Rekonfigurierbarkeit der Adaption). Der Adaptionskontext ist daher durch die Kontextelemente gegeben, welche das Metamodell des K-Modells abbilden. In 6.4.1 wurde nämlich festgelegt, dass für jedes Element des K-Modells außer den Kontextelementen, die sich selbst beschreiben können, jeweils ein eigenes Kontextelement existiert (z.B. Actuator1ModelKontext für Aktuator1 aus Abb. 52), das eine Beschreibung des Elementes als XML Dokument enthält (6.3). Diese Beschreibungen werden vom Modellaktuator (der `Activator` Komponente) dazu verwendet, zur Laufzeit gemäß dem momentanen Stand der Spezifikation im Kontext, Dienste analog zu den in der Spezifikation genannten Sensoren, Interpretern, Aktuatoren und Kontextelementen hinzuzufügen (zu aktivieren), zu entfernen (zu deaktivieren) oder ihre Bindung an die diensterfüllenden Komponenten (Realisierung/Implementierung) zu verändern. Für den Entwurf einer Kalibrierung (Adaption der Adaption) müssen daher lediglich weitere Sensoren und Interpreter hinzugefügt werden, welche die Beschreibungen der Elemente der regulären Adaptionen verändern. Natürlich ist es auch möglich, wiederum die Beschreibungen der Sensoren und Interpreter der Kalibrierung zu verändern (mehrfache Anwendung der Kalibrierung auf sich selbst).

Für den Entwurf einer konkreten Kalibrierung bietet sich aufgrund der Natur des Frame-Problems, das auf Modellierungsannahmen beruht, die ihre Gültigkeit verlieren (siehe 3.2.1), ein methodisches Vorgehen auf Basis der Abschätzung der Änderungswahrscheinlichkeit der Modellierungsannahmen für die Kontextadaption an.

Modellierung impliziter Annahmen

Für jedes Kontextelement der initialen Kontextadaption einschließlich der Metamodellelemente wird dabei im ersten Schritt versucht, die enthaltenen impliziten Annahmen als explizite Annahmen zu formulieren.

Definition 23: *explizite Annahme*, *Kontextkonstante*

Kontextkonstanten sind Kontextelemente ohne Eingabe (Sensor, Interpreter). Sie repräsentieren eine statische explizite Annahme des Modells.

Normalerweise ist die Aufgabe, alle impliziten Annahmen einer Modellierung explizit zu beschreiben ein informeller Prozess mit extremen Schwierigkeiten bei der Erzielung und Überprüfung der Vollständigkeit. Mit anderen Worten, es handelt sich um eine unrealistische Aufgabe, sofern es sich um ein realistisches System und nicht um ein künstlich eingeschränktes Labor- oder sogenanntes Toy-System [78] handelt. Die Vollständigkeit ist in diesem Fall zum Glück aber nicht notwendig. Die Kalibrierung war ja nämlich gerade zu dem Zweck konzeptioniert worden, den negativen Auswirkungen einer unvollständigen Modellierung impliziter Annahmen entgegenzuwirken. Aus diesem Grund war auch für das Teilsystem der Kontextadaption (einschließlich der Kalibrierungsadaptionen) die Fähigkeit zur Selbstkalibrierung bzw. vollständigen Rekonfiguration gefordert worden. Dennoch ist es aber sinnvoll die wichtigsten impliziten Annahmen explizit zu machen, weil dadurch der Kalibrierungsprozess strukturiert und damit auch für den Benutzer übersichtlicher und einfacher gestaltet werden kann.

Aus dem gleichen Grund erlauben auch nichtubiquitäre Open Source Programme die Anwenderkonfiguration bestimmter häufig zu ändernder Verhaltensaspekte über eine Benutzerschnittstelle, obwohl die gleichen Anpassungen und noch viele mehr über eine direkte Modifikation des Quelltextes erreicht werden könnten.

Das verwendete Modell der Kontextadaption leistet bereits bei der Analyse solcher endbenutzergeeigneten Kalibrierungshilfen eine gewisse Hilfestellung, besonders was die Identifizierung von impliziten technischen Annahmen angeht. Kontextelemente, die keine Konstanten sind und nicht zum Metamodell gehören, enthalten nämlich in der Regel keine implizite Annahmen, ausgenommen über ihre eigene Relevanz. Diese ist aber bereits durch die Existenz des Elementes selbst ausgedrückt.

Diese Behauptung kann man sich durch folgende Überlegung plausibel machen. Kontextelemente sind vergleichbar mit Variablen. Sie stellen damit keine Modellierungsannahme dar, sondern das genaue Gegenteil. Implizite Annahme verstecken sich höchstens in der Art der Repräsentation (Datenformat etc.) und der Tatsache ihrer Existenz.

Diese Behauptung kann man sich durch folgende Überlegung plausibel machen. Kontextelemente sind vergleichbar mit Variablen. Sie stellen damit keine Modellierungsannahme dar, sondern das genaue Gegenteil. Implizite Annahme verstecken sich höchstens in der Art der Repräsentation (Datenformat etc.) und der Tatsache ihrer Existenz. Kontextelemente können jedoch auch untypisierte Informationen speichern, bzw. sind indirekt abhängig von den mit ihnen verbundenen Elementen (Sensoren etc.). Sie brauchen daher für die Kalibrierung nicht berücksichtigt werden, solange sie mit einem Sensor verbunden, d.h. keine Konstanten sind.

Kritischer als die einfachen Kontextelemente sind Kontextkonstanten (also auch die expliziten Annahmen selbst) und als Teilmenge davon auch die Kontextelemente, die das Metamodell bilden, solange sie noch nicht mit Sensoren oder Ähnlichem verknüpft, also kalibrierbar sind.

Die nahe liegende implizite Annahme, die sich hinter einer Kontextkonstante verbirg ist zwar lediglich, dass es sich um einen unveränderlichen Wert handelt. Diese Annahme stützt sich aber normalerweise auf eine Reihe weiterer Annahmen, für die, sobald explizit modelliert (als Kontextkonstanten) wieder die gleiche Aussage gilt. Beispiele für solche Annahmen sind:

- Das System ist nur einen bestimmten Zeitraum im Einsatz.
- Es wird nur in der vorgesehenen Weise verwendet.
- Nur für einen bestimmten Nutzerkreis, Kulturkreis oder geografischen Bereich.
- Änderungen sind innerhalb der Messgenauigkeit.
- Ein bestimmter Sensor ist immer verfügbar, liefert immer korrekte Werte usw.

Zusammengefasst lässt sich also festhalten, dass eine Kontextkonstante eine explizite Annahme darstellt, die sich auf eine potenziell unendliche Menge von weiteren Annahmen abstützt. Die meisten Annahmen beziehen sich auf die Metamodell-Kontextelemente. So enthält beispielsweise der Metamodellkontext eines immer an die gleiche Kernsystemkomponente gebundenen Sensors ohne Änderungsmöglichkeit die immer gleiche Sensorspezifikation. Es handelt sich also um eine explizite Annahme, deren Unveränderbarkeit zudem in viele weitere explizit zu machende implizite Annahmen aufgespalten werden kann.

Abschätzung der Änderungswahrscheinlichkeit

Für Entwurfszwecke muss eine solche Auffächerung von Annahmen aber lediglich so tief fortgeführt werden, bis eine Abschätzung der Änderungswahrscheinlichkeit möglich ist (Abb. 53).

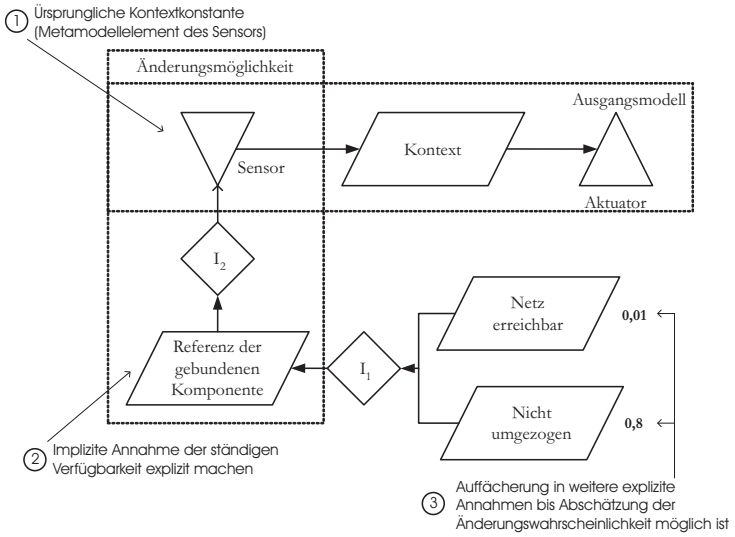


Abb. 53: Beispiel Änderungsmöglichkeit einer Sensordefinition

Definition 24: *Änderungswahrscheinlichkeit* einer Annahme _____

Die Wahrscheinlichkeit, dass eine Annahme über einen angenommenen Zeitraum und oder die Anzahl der Installationen falsch wird.

Welcher Zeitraum und ob ein einzelnes System oder alle Installationen als Basis der Wahrscheinlichkeitsabschätzung verwendet wird, kann nicht sinnvoll pauschal festgelegt werden, sondern hängt von der Art der Anwendung und den durch Kontextadaption ausgelösten Aktionen sowie den Folgen einer Fehleinschätzung ab. Häufig verwendbare Basiszeiträume sind eine bestimmte Betriebsdauer der Anwendung oder die gesamte Lebensdauer.

Liegen abhängig von der Art ihrer Verknüpfung ein oder mehrere Ausgangsannahmen einer abgeleiteten Annahme über einem Schwellwert, kann die abgeleitete Annahme als ein Kandidat für Kalibrierung (Abb. 53) markiert werden. Dies gilt insbesondere für die Metamodellelemente der Listen von Elementtypen (Sensor, Interpreter, Aktuator, Kontext). Solange diese nicht mit einem Sensor oder Interpreter als Änderungsmöglichkeit versehen sind, kann die Kontextadaption bezüglich dieses Elementtyps nicht durch Hinzufügen oder Entfernen kalibriert werden. Die jeweilige Liste modelliert in die-

sem Fall die explizite Annahme, dass dies zur Laufzeit (hier ist der Basiszeitraum die Lebensdauer) nicht notwendig ist (Abb. 54).

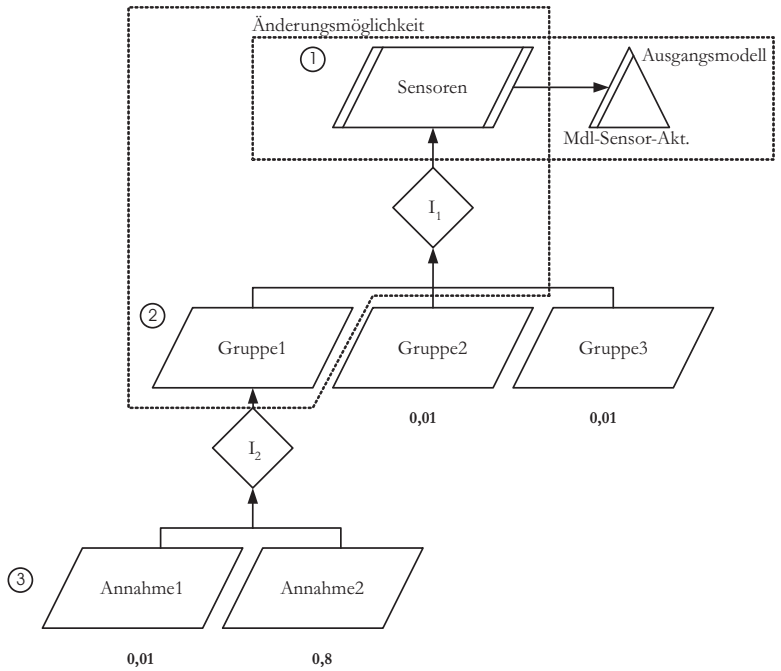


Abb. 54: Beispiel für Hinzufügen und Entfernen einer Gruppe von Sensoren.

Abb. 54 zeigt ein Beispiel, wie sich die Stabilitätsannahme einer solchen Modellelementliste wie zuvor beschrieben, in weitere Annahmen unterteilen oder bezüglich ihrer Änderungswahrscheinlichkeit abschätzen und als Kalibrierungsmöglichkeit identifizieren lässt.

Die Kontextkonstante im Ausgangsmodell bedeutet in diesem Fall die ursprüngliche Annahme, dass ein Hinzufügen neuer Sensoren nicht notwendig ist (Abb. 54 Nr. 1).

Diese Annahme lässt sich zunächst differenzierter nach unterschiedlichen Sensorgruppen betrachten, von denen einige als stabil abschätzbar sind (Abb. 54 Nr. 1).

Die Stabilität der ersten Gruppe basiert dagegen auf einer instabilen Annahme. Daher kann in diesem Entwurfsschritt die Ableitung dieser instabilen Annahme als Kalibrierungsmöglichkeit markiert werden (Abb. 54 Nr. 3).

Manchmal ist eine quantitative Abschätzung von Änderungswahrscheinlichkeiten nicht möglich, beispielsweise weil keine statistischen Erfahrungswerte

aus vorangegangenen Projekten vorliegen. In solchen Fällen lässt sich die Methode aber auch mit einer qualitativen Einschätzung (gering, mittel, hoch etc.) verwenden.

Auswahl eines geeigneten Schwellwertes

Die Verwendung qualitativer Abschätzungen ist möglich, da für die Entscheidung, ob eine explizit modellierte Annahme besser durch eine Kalibrierungsmöglichkeit ersetzt wird, nicht alleine die gerade eben beschriebene Änderungswahrscheinlichkeit, sondern vor allem der ausgewählte Schwellwert entscheidend ist. Diese Behauptung folgt aus der Feststellung, dass für die Wahrnehmung von SUB, die ja durch unrichtig werdende Modellierungsannahmen entstehen, auch der Perzeptionsfaktor eine wesentliche Rolle spielt (siehe auch Verstärkungseffekte in 3.2.4). Es ist also vernünftig, auch beim Entwurf der Gegenmaßnahmen (Kalibrierung) Perzeptionsfaktoren in Form von unterschiedlichen Schwellwerten zu berücksichtigen. Theoretisch bestünde zwar auch die Möglichkeit, immer den kleinsten Schwellwert anzuwenden also generell immer von einer hohen Perzeption auszugehen oder im Zweifelsfall jede Annahme in eine Kalibrierungsmöglichkeit zu verwandeln. Dadurch steigt aber der technische Aufwand erheblich. Oft lassen sich auch nicht für alle Annahmen automatische Sensoren realisieren. Dies führt dazu, dass der Teil der Kalibrierung, der auf direkter Interaktion mit dem Benutzer basiert erheblich an Komplexität gewinnt. Dies geht bis zu einem Punkt, wo sie für den Endbenutzer nicht mehr beherrschbar oder zu aufwändig und damit unbrauchbar wird, weil dadurch die Ubiquitätsbedingungen verletzt werden.

Folglich macht es Sinn, für einzelne Annahmen über deren Kalibrierbarkeit anhand individueller Schwellwerte zu entscheiden. Ein jeweils geeigneter Schwellwert kann über einen Störfaktor ermittelt werden. Der Störfaktor wurde empirisch aus der Fallstudie (Kapitel 2) gewonnen und basiert auf einer Abschätzung der Perzeption und Folgen einer unerwünschten Auslösung einer Adaption.

Definition 25: *Störfaktor*

Der Störfaktor einer unerwünschten Auslösung oder nicht Auslösung einer Adaption setzt sich zusammen aus einem Grundfaktor (b) der Aktion, der Häufigkeit der Auslösung (h), einen Gewöhnungsfaktor (g) und der Regelmäßigkeit (r) der Auslösung, beispielsweise

$$S \overset{def}{=} (b \quad g \quad r) \quad h / (1 \quad g \quad r \quad h).$$

Die angegebene Formel wurde empirisch für die Anwendungsdomäne der Fallstudie gewonnen. Andere Gewichtungen sind ebenfalls denkbar. Zu beachten ist, dass der Faktor *b* das Maximum der grundlegenden Perzeption einer Aktion beschreibt. Jede Aktion hat mindestens zwei Perzeptionswerte, welche der Auslösung und der Nicht-Auslösung entsprechen. Parametrisierte Aktionen haben unter Umständen mehr als zwei Perzeptionsfaktoren.

Fkt	Bereich	Art der Ermittlung/Abschätzung
b	[0..1]	Prozentualer Anteil der Testbenutzer, die das erste Auftreten einer Fehlauslösung oder Nichtauslösung als störend bewerten (würden).
h	[0..n]	Anzahl reguläre Aktivierungen pro Tag eines Testlaufes
g	[0..1]	Absolutdifferenz zwischen b und dem Anteil der Testnutzer, die eine Fehlauslösung nach mehreren Wiederholungen noch als störend bewerteten.
r	[0..1]	Wahrscheinlichkeit, die Aktivierung aus Sicht des Benutzers vorhersagen zu können.

Mithilfe des Störfaktors einer Aktion kann für alle Annahmen, von denen die Auslösung einer Aktion abhängig ist, ein geeigneter Schwellwert festgesetzt werden. Damit ergaben sich für die adaptiven Aktionen der Fallstudie zum Beispiel die folgenden Störfaktoren:

Aktion	Berechnung	S
Licht an	b=0,6 h=10 g=0,3 r=0,8	2,5
Licht aus	b=1,0 h=10 g=0,3 r=1,0	3,3
Lüfter an	b=0,3 h= 1 g=0,0 r=1,0	0,3
Lüfter aus	b=0,6 h= 1 g=0,0 r=0,0	0,6
Zeige Zugverspätung	b=0,0 h= 5 g=0,3 r=0,5	0,43
Zeige Wetter	b=0,0 h= 5 g=0,3 r=0,5	0,43

Für den Prototyp der Fallstudie wurden für den Entwurf der Kalibrierung alle Änderungswahrscheinlichkeiten als gering angenommen (qualitative Abschätzung). Die Entscheidung für oder gegen eine Kalibrierung wurde daher alleine aufgrund des Störfaktors der jeweiligen Adaption getroffen. Überschritt dieser 1.0, so wurde die qualitative Abschätzung "gering" als Schwellwert gewählt. Diese Festlegung ergab sich aus der Überlegung, dass Aktionen mit kleinerem Störfaktor (z.B. der Lüfter mit 0,6) einfach zu selten aktiviert werden, um in Fällen mit geringer Änderungswahrscheinlichkeit den Entwic-

klungsaufwand der Kalibrierung dafür zu rechtfertigen. Hätte es im untersuchten System Modellierungsannahmen mit qualitativ höherer Änderungswahrscheinlichkeit gegeben, wäre für solche Fälle ein anderer Schwellwert (z.B. "hoch") als Kalibrierungsgrenze festgelegt worden. Aktionen mit einem Störfaktor <0,5 wurden dagegen bei den Versuchen überhaupt nicht als SUB wahrgenommen. In solchen Fällen kann dann auch unabhängig von der Änderungswahrscheinlichkeit auf eine Kalibrierung verzichtet werden (Schwellwert).

Enthält eine Funktion mehrere Adaptionen, kann sie mehrere unterschiedliche Störfaktoren besitzen.

In diesem Zusammenhang ist aber anzumerken, dass von einem unterhalb der Perzeptionsschwelle liegenden Störfaktor einer einzigen Adaption nicht automatisch auf die SUB-Robustheit der betroffenen Funktion geschlossen werden kann. Einige Funktionen des Prototyps aus 2.3 mit SUB-robusten Adaptionen besitzen nämlich mehrere Adaptionen, die sich in ihrem Störfaktor sehr stark unterscheiden können:

> Ein besonders deutliches Beispiel ist die Verspätungsinformationen des öffentlichen Nahverkehrs. Die Adaption zur Anzeige der Information ist mit einem Störfaktor von <0,5 für die gegebene Anwendungsdomäne SUB-robust. Das bedeutet, der Benutzer ist aufgrund der leichten Ignorierbarkeit tolerant gegenüber Anzeigen von Verspätungen, auch wenn er gerade nicht die Benutzung eines Verkehrsmittels plant. Wegen der bekannt schwierigen Vorhersagbarkeit ist auch noch die Nicht-Anzeige tolerierbar. Die richtige Anzeige falscher Informationen (zu geringe oder zu hohe Verspätungszeit im Kontext) besitzen dagegen einen höheren Störfaktor und sollten daher kalibrierbar sein, beispielsweise durch den Wechsel des Anbieters oder durch die Einrechnung personalisierbare Sicherheitsfaktoren.

7.3.2 Entwurfsprinzipien der Benutzerschnittstelle für Kalibrierung

Die im letzten Schritt entworfenen Kalibrierungsmöglichkeiten auf der Basis von Modellierungsannahmen, deren Änderungswahrscheinlichkeiten und perzeptionsbasierten Schwellwerten stellen noch keine vollständige Kalibrierung dar. Vielmehr handelt es sich um eine Kalibrierungsschnittstelle, das heißt eine Festlegung, welche Änderungen auf Basis welcher Kontextinformationen zulässig sind. Betrachtet man die Kalibrierung als Adaption der Adaption, wurde mit den Kalibrierungsmöglichkeiten also lediglich der Adaptionskontext einer Adaption definiert, welche das Teilsystem der Kontextadaption verändert.

Gemäß der allgemeinen Methodik zum Entwurf von Kontextadaptionen würde im nächsten Schritt nun die Adaptionslogik sowie ein geeigneter Sen-

sorkontext festgelegt (7.2.2). Für die Kalibrierung bedeutet dies die Einbettung der in 7.2.3 entworfenen Kalibrierungsschnittstelle in eine Benutzerschnittstelle der Kalibrierung, beziehungsweise den Entwurf von mehrfachen Anwendungen der Kalibrierung auf sich selbst (Automatisierung, Lernfähigkeit), mit dem Ziel, die Benutzung der Kalibrierung so weit wie möglich ubiquitär zu gestalten. Wichtig dabei war lediglich, dass aufgrund der Natur des Frame-Problems am Ende der Mehrfachanwendungen eine direkte und bewusste Interaktion mit einem menschlichen Benutzer steht (siehe tertiäre Interaktion in Abb. 13). Diese direkte Interaktion kann aber ebenfalls in Form eines Sensorkontextes im Modell der Kontextadaption abgebildet werden (Emulationseigenschaft der Kontextadaption 5.1.1).

Im Entwurf wird die Benutzerschnittstelle der Kalibrierung dabei in drei Teilbereiche unterteilt, die sich nach den verschiedenen Aktionen des Benutzers zur Verhinderung (Kalibrierung) von SUB-Phänomenen richten:

- Die Ausgabe (Dokumentation) der gegenwärtig gültigen Kontextadaptionen, einschließlich des momentanen Kontextes. (*Verstehen*)
- Das Suchen und das Überprüfen der Ursachen bereits aufgetretener beziehungsweise antizipierter Ursachen eines SUB (vgl. Debugging) in gegenwärtigen oder vergangenen Kontextadaptionen. (*Erkennen*)
- Die Eingabe einer veränderten Kontextadaption, die das SUB beheben soll. (*Verändern*)

Die Entwurfsschritte für jede dieser drei Teilschnittstellen orientieren sich dabei an dem häufig in Zusammenhang mit der Entwicklung von Benutzerschnittstellen verwendeten MVC [88] Entwurfsmuster. Für mithilfe des Framework aus Kapitel 6 entwickelte Anwendungen ist das Modell (*model*) der in 7.3.1 als Teil der Kalibrierungsschnittstelle definierte Kalibrierungskontext. View und Controller des MVC Patterns müssen durch zusätzliche Entwicklungsschritte für jede Teilschnittstelle konstruiert werden:

- Festlegen einer uni- oder multimodalen Interaktionsform und die Transformation der einzelnen Elemente eines K-Modells oder des jeweiligen Kalibrierungskontext-Inhaltes in Darstellungen des jeweiligen Mediums. (Präsentation)
- Spezifikation der Präsentationslogik, wiederum in Form einer Kontextadaption. (Kontrolle)

Der als Modell (*model*) der Benutzerschnittstelle genutzte Kalibrierungskontext enthält in den meisten Fällen die Beschreibung eines K-Modells oder eines vereinfachten Ausschnittes davon (siehe 6.4.1). In manchen Fällen kann diese Zuordnung allerdings sehr abstrakt definiert sein, beispielsweise wenn

Die Benutzerschnittstelle der Kalibrierung wird als Kontextadaption abgebildet, um die Ubiquitätsbedingungen zu erhalten.

die Betätigung eines Schalters eine ganze Gruppe von Adaptionsaktionen deaktivieren soll. Für die Präsentation (*view*) kann es daher notwendig sein, die Visualisierung nicht auf den Kalibrierungskontext selbst, also beispielsweise den Schalter, sondern auf dessen Wirkung, also die deaktivierten Aktionen zu beziehen. Die Präsentation selbst erfolgt durch Aktuatoren respektive Sensoren, welche die dort abgelegten Beschreibungen in eine für das Interaktionsmedium geeignete Darstellung überführen (Grafik, Sprache etc.) und umgekehrt.

Die Kontrolllogik der Benutzerschnittstelle (*control*) ist in der Regel ebenfalls bereits über den Kalibrierungskontext spezifiziert. Komplexe Benutzerschnittstellen, insbesondere bei der Zusammenführung multimodaler Ein- und Ausgabeformen können es jedoch notwendig machen, eine komplexe Interaktionslogik über zusätzliche Interpreter zu realisieren, die auf dem Kalibrierungskontext basieren.

Für die drei Teilschnittstellen (Verstehen, Erkennen, Ändern) gestalten sich die Entwurfsschritte für die Modellanbindung und Kontrolllogik der Benutzerschnittstelle für Kalibrierung je nach Ausprägung leicht unterschiedlich. Daher wird im Folgenden kurz auf die jeweiligen Besonderheiten eingegangen.

Ausgabe und Dokumentation von Kontextadaption

Der Ausgangspunkt für jede Kalibrierung durch den Benutzer ist dessen Information über den gegenwärtigen Zustand der Adaption (das momentan zu erwartende Anwendungsverhalten) sowie davon ausgehend mögliche Adaptionen bei einer Änderung der Anwendungssituation. Auf diese Weise können bereits viele SUB-Phänomene verhindert werden, indem unerfüllbaren Erwartungen des Benutzers vorgebeugt wird. Zumindest aber erlaubt es dem Benutzer, bis zu einem gewissen Grade kritische Situationen vorherzusehen, in denen SUB-Phänomene auftreten könnten.

> Ist dem Benutzer der Anwendung aus der Fallstudie (Kapitel 2) beispielsweise (aus dem Benutzerhandbuch) bekannt, dass die Steuerung der Beleuchtung im Wesentlichen von einer Bewegungserkennung abhängt, könnte er vorhersehen, dass längeres regungsloses Verharren in einem Raum möglicherweise zu einem SUB führen könnte.

Leider verträgt sich der Anspruch der umfassenden Information nicht immer mit den Ubiquitätsbedingungen. Im Entwurf einer Benutzerschnittstelle muss dieser Umstand daher berücksichtigt werden. Man unterscheidet des-

halb zwischen zwei Arten der Ausgabe, d.h. Dokumentation von Kontextadaption.

Unter den *benutzerinitiierten* Ausgabeformen fasst man alle Arten der Information des Benutzers zusammen, die auf dessen Wunsch (bewusst) und in exklusiver Interaktion (sozusagen außerhalb der Ubiquität) erfolgen.

Benutzerinitiierte Ausgabe bei konkretem Verdacht des Benutzers auf SUB.

Benutzerinitiierte Ausgaben sind beispielsweise das Lesen eines gedruckten Manuals, einer Online-Hilfe des Systems oder die Aktivierung eines Editors für die Kalibrierung.

In der Regel wird der Benutzer von dieser Form der Kalibrierung Gebrauch machen, wenn bereits ein SUB aufgetreten ist, oder der Benutzer aufgrund anderer Informationsquellen bereits eine konkrete Vermutung in Hinblick auf ein mögliches zukünftiges Auftreten hat.

Der Benutzer weiß zum Beispiel aus Erfahrung, dass immer dann mit einem SUB zu rechnen ist, wenn neuartige oder unregelmäßige Ausnahmesituationen eintreten (eine Feier, Renovierungsarbeiten etc.). Mithilfe des Handbuches könnte er sich vorher vergewissern, mit welchen Adaptionen er in solchen Fällen zu rechnen hat.

Wichtig zu bemerken ist, dass die Ubiquität des Gesamtsystems trotz der exklusiven Interaktion erhalten bleibt. Während der Benutzer die Ausgaben studiert, können nämlich durchaus weitere Adaptionen in seinem Sinne im Hintergrund ablaufen. Lediglich die Kalibrierung selbst ist nicht mehr ubiquitär, das heißt ortsunabhängig und gleichzeitig neben anderen Tätigkeiten nutzbar.

Für den Prototyp der ubiquitären Smarthome Anwendung aus der Fallstudie in Kapitel 2 wurden zwei unterschiedliche benutzerinitiierte dokumentierende Kalibrierungsschnittstellen konzipiert. Das erste Beispiel (Abb. 55) zeigt eine benutzerinitiierte Kalibrierungsausgabe in Form einer Online-Hilfe. Diese erlaubt allerdings keine Änderungen am Adaptionsverhalten. Es handelt sich daher um eine unidirektionale Kalibrierungsinteraktion, die im Wesentlichen der Vorauserkennung von SUB dient. Dem Benutzer bleibt aber die Möglichkeit, dem Eintreten eines erkannten SUB durch Änderung seines eigenen Verhaltens beziehungsweise seiner Erwartungshaltung zu begegnen, oder auf eine andere Kalibrierungsschnittstelle zu wechseln, welche auch Änderungen zu lässt. Abb. 59 (rechts) zeigt dazu die zweite benutzerinitiierte Kalibrierungsausgabe, die Teil eines Editors ist, der auch Änderungen zulässt.

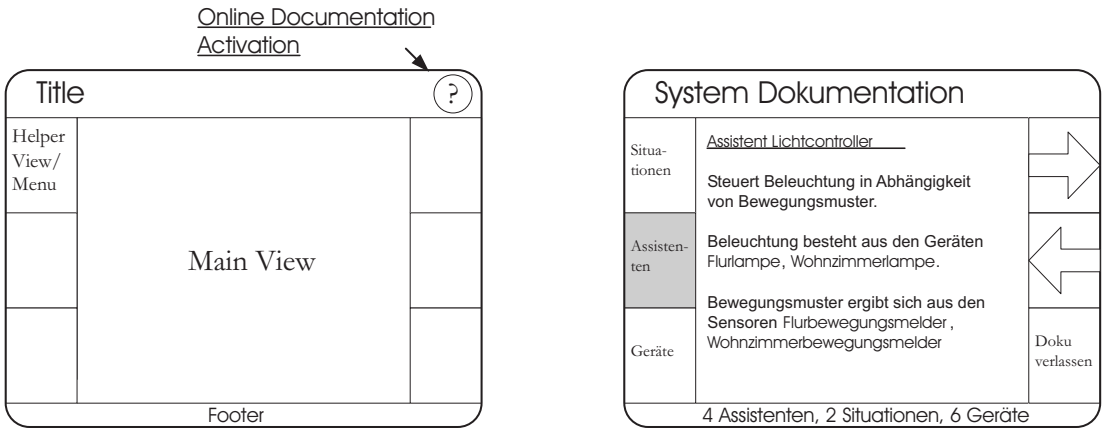


Abb. 55: Beispiel einer Kalibrierungsausgabe der Fallstudie (Dokumentation)

Im Gegensatz zu den benutzerinitiierten Kalibrierungsausgaben können die *systeminitiierten* Ausgabeformen die Ubiquitätsbedingungen auch bezüglich der Kalibrierung erfüllen. Dies bedeutet aber nicht, dass dabei keine direkten Interaktionen involviert sind. Da das System aber die Kontrolle über die Initiierung besitzt, ist es in der Lage, die Ausgabe an die Bedürfnisse des Benutzers anzupassen. Allerdings ist das System normalerweise nicht in der Lage, SUB im Voraus zu erkennen. Die meisten systeminitiierten Ausgabeformen müssen sich daher auf unbewusste Interaktionen im Hintergrund beschränken. Zu diesem Zweck muss die Beschreibung von gerade aktiven und möglichen Adaptionen (siehe 6.3 oder die grafische Repräsentation 6.4) allerdings in unbewusst oder nebenbei wahrnehmbare Signale transformiert werden. Abschnitt 7.3.4 geht noch einmal näher auf den Entwurf solcher Techniken ein.

Eine andere Möglichkeit ist die Einbindung heuristischer Methoden zur Erkennung kritischer Situationen als weitere Kontextadaptionen. Diese beobachten den Kontext und versuchen daraus Situationen zu erkennen, in denen ein SUB aufgetreten und möglicherweise unbemerkt geblieben sein könnte. Alternativ können sie eine Adaption in einer gegenwärtigen Situation verzögern, bis der Benutzer Zeit findet, die Situation anhand einer benutzerinitiierten Ausgabe zu beurteilen.

Eine Kontextadaption könnte beispielsweise die anderen Adaptionen und deren Kontext dahingehend überwachen, ob eine ungewöhnliche Belegung oder eine bisher nie durchgeführte Adaption aktiviert werden soll. Falls möglich können die sich daraus ergebenden Aktionen so lange verzögert werden, bis ein bestätigendes Eingreifen des Nutzers möglich ist.

Solche Ausgabeformen der Kalibrierung sind aber mit Vorsicht zu genießen, da sie sehr leicht mehr SUB erzeugen können, als sie verhindern helfen. Man verwendet sie daher oft nur in abgeschwächter und sehr spezialisierter Form.

Einzelne Adaptionen, deren ungewollte Ausführung schwerwiegende Folgen haben könnte, werden mit einer speziellen Kalibrierung in Form einer Rückfrage versehen. Diese direkte Interaktion stört nicht unbedingt die Ubiquität, da sie auch auf für den Nutzer günstige Augenblicke verschoben werden kann, wie etwa die Bestätigung einer größeren Bestellung.

In der Fallstudie aus Kapitel 2 haben sich besonders Heuristiken als nützlich erwiesen, die auf der Basis der Analyse redundanter Informationen arbeiteten. Die Wirksamkeit beschränkte sich aber im Wesentlichen auf SUB-Phänomene der Kategorie "unbemerkte Ressourcenänderungen" und "Messungenauigkeiten".

In 2.4 war beschrieben worden, dass in ubiquitären Anwendungen SUB einer bestimmten Kategorie entstehen kann, weil die Anwendung in der Regel nicht zwischen dem Defekt und der Abwesenheit einer Ressource unterscheiden kann, während der Benutzer zwar oft die Anwesenheit, nicht aber den Defekt eines Gerätes erkennt.

Für die in 2.4 genannten Beispiele solcher SUB konnte zumindest für den Fall nichtmobiler Geräte Abhilfe durch eine Ausgabekalibrierung auf Basis einer Defektheuristik erzielt werden. Grundlage dafür war die Erkennbarkeit der Geräte unabhängig von ihrer Funktionstüchtigkeit. Dies konnte realisiert werden, indem zu jedem Gerät, auch wenn es automatisch erkennbar war, ein zusätzliches Steckmodul in eine Konfigurationskonsole (vergleichbar einem Sicherungskasten) gesteckt wurde (siehe Abb. 56). Dieses Steckmodul enthält die Spezifikation einer oder mehrere Sensoren/Aktuatoren des zugehörigen Gerätes. Die Defekterkennung ist als Interpreter realisiert, der eine, bezogen auf die bisherigen Werte überdurchschnittlich lange Abwesenheit neuer Sensordaten bei gleichzeitiger Anwesenheit des Steckmoduls als mögliche SUB Gefahr signalisiert. Die Signalisierung erfolgt durch eine systeminitiierte Ausgabekalibrierung, beispielsweise das Blinken des Steckermoduls oder eine Email-Nachricht an den Benutzer.

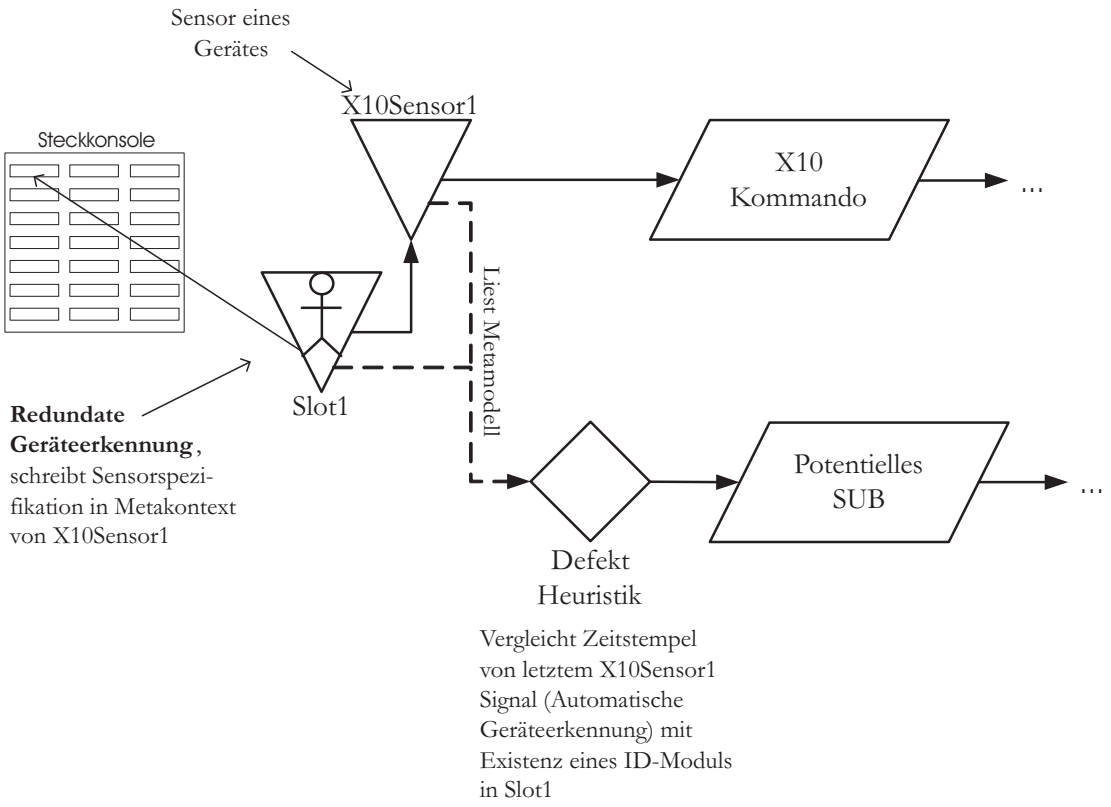


Abb. 56: Beispiel einer systeminitiierten Kalibrierungausgabe

Eingabe veränderter Kontextadaption

Als Gegenstück der Kalibrierungsausgabe stehen für die Eingabe geänderter Kontextadaption analog der Ausgabe wiederum zwei grundlegende Möglichkeiten zur Verfügung.

Die *aktive Kalibrierung* ist gekennzeichnet durch eine bewusste Interaktion zwischen Benutzer und System mit dem Ziel, das Adaptionsverhalten an die besondere Situation des Benutzers anzupassen. Dieser Kalibrierungsvorgang wird im K-Modell mittels Mediatoren abgebildet, welche die Benutzerschnittstelle der aktiven Kalibrierung abbilden (siehe Abb. 57).

Wie für die benutzerinitiierte Kalibrierungsausgabe bereits beschrieben, ist die Ubiquität der Kalibrierung (nicht der Anwendung) in solchen Fällen direkter Interaktion eingeschränkt. Die Kalibrierung kann also höchstens in gewissen zeitlichen Abständen stattfinden.

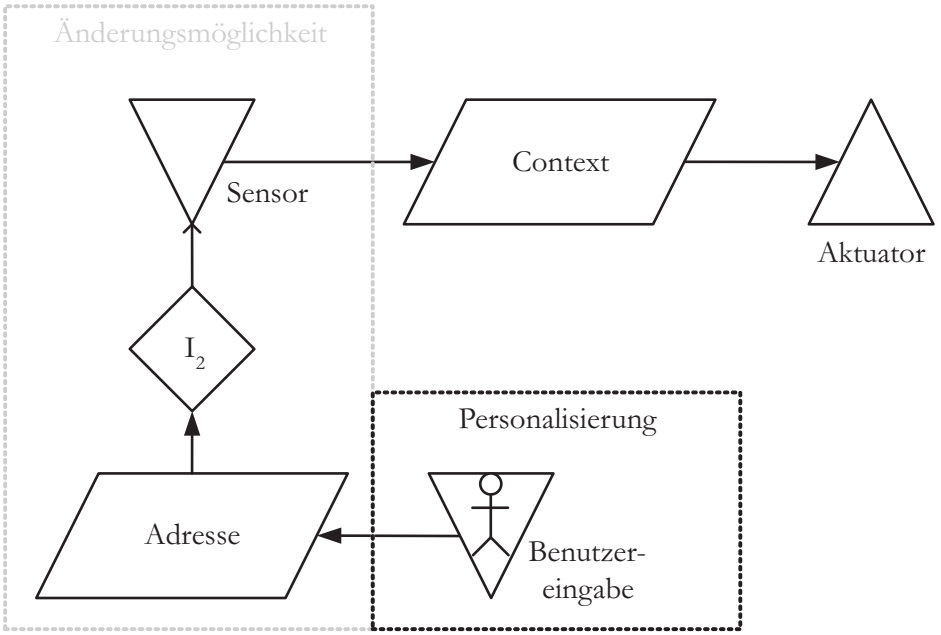


Abb. 57: Eingabe von Änderungen durch aktive Kalibrierung

Abb. 57 zeigt einen Fall, in dem nur für einen spezifischen Benutzer relevante Sonderfälle durch Verändern der Sensorbeschreibung oder Hinzufügen zusätzlicher Sensoren kalibriert werden:

Für ein einfaches Beispiel kann wieder die Kontextinformation über die Temperatur eines Raumes herangezogen werden. Es wird die Temperatur in °C benötigt. Leider existiert im Beispiel diesmal auch kein Sensor in Fahrenheit. Ist im initialen Kontext für den betroffenen Sensor eine Änderungsmöglichkeit des K-Modells spezifiziert worden, die für aktive Kalibrierung genutzt wird, kann ein Benutzer für die Behebung dieses Problems auch nur für ihn gültige Sonderfälle berücksichtigen. So könnte er in diesem Fall veranlassen, den Temperatursensor eines Nebenraumes zu verwenden, der durch einen Durchgang ohne Türe verbunden ist und in dem normalerweise die gleiche Temperatur herrscht, wie im Nebenraum.

Es lassen sich aber nicht nur die Beschreibungen existierender Modellelemente durch die aktive Kalibrierung verändern, sondern es können auch neue Elemente hinzugefügt, oder entfernt werden (siehe Abb. 58).

Die Modellierung dieses K-Modellausschnitts (Abb. 58) zeigt auch gleichzeitig die Realisierung von einfachen Sicherheitsmechanismen für die Modifizierungen an der Modellierung der Kontextadaption zur Laufzeit. Ein Benutzer-

dialog oder grafischer Editor dient als Sensor für konfigurierte Änderungen am Kontextmodell.

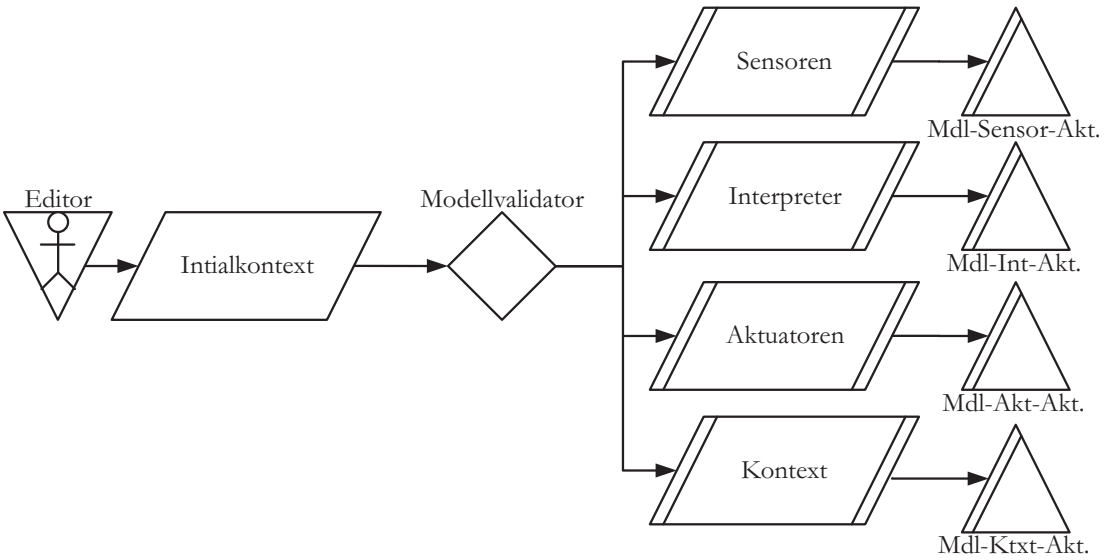


Abb. 58: Schema der Kalibrierung eines K-modells

Diese werden von einem Validator überprüft, bevor die entsprechenden Änderungen in das Metamodell geschrieben und von den Modellaktuatoren implementiert werden. Der Modellvalidator enthält die notwendigen Sicherheitsüberprüfungen und Konsistenzprüfungen und sollten deshalb als eine nicht (oder nur über einen gesicherten Mechanismus) austauschbare Komponente der modellierten Anwendung spezifiziert werden.

Für den Prototyp der ubiquitären Smarthome Anwendung aus der Fallstudie in Kapitel 2 erfolgt die aktive Kalibrierung aus Sicherheitsgründen im Allgemeinen ausschließlich über eine spezielle Kalibrierungskonsole (Abb. 59). Diese ist vergleichbar mit einem Schalt- oder Sicherungskasten. Neue Sensoren, Assistenten und Aktuatoren werden ausschließlich über Steckmodule (USB Speichersticks) dem System hinzugefügt (Abb. 59 links), oder temporär durch Umstecken auf einen der Programmierplätze für eine Kalibrierung über das zugeordnete Monitor Terminal (Abb. 59 rechts) oder ein Software-Update freigegeben.

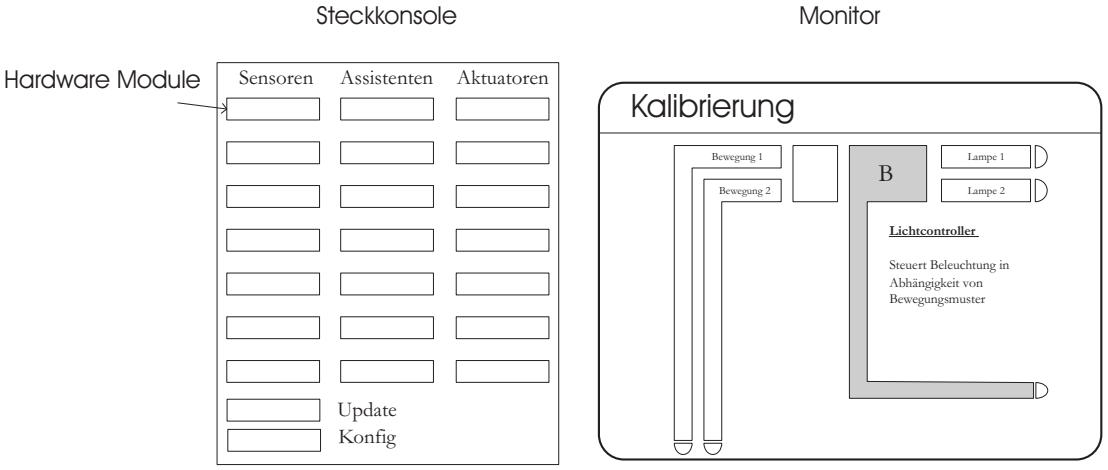


Abb. 59: Beispiel einer aktiven Kalibrierung mit Hardware Sicherung

Eine Alternative zur aktiven Kalibrierung stellt die *passive Kalibrierung* dar. Bei dieser finden die Änderungen der Adaption nicht über bewusste direkte Interaktionen zwischen System und Benutzer statt. Das bedeutet aber nicht, dass es keine direkten Interaktionen gibt, sondern lediglich, dass dem Benutzer nicht bewusst ist, dass damit eine Kalibrierung verbunden ist. Passive Kalibrierung findet also immer dann statt, wenn das System durch Beobachtung des Nutzerverhaltens auf dessen Präferenzen oder Bedürfnisse schließt.

In der Modellierung einer Kontextadaption kann die passive Kalibrierung dadurch von der aktiven unterschieden werden, dass bei ersterer keine Mediatoren direkt in die Kalibrierung involviert sind.

Abb. 60 zeigt ein Beispiel, in dem die Kalibrierung selbst keine Mediatoren enthält. Die zu kalibrierende Kontextadaption dagegen schon (als Teil der normalen Adaption, nicht der Kalibrierung!). Die Kalibrierung selbst erfolgt unbewusst. Das System beobachtet den Benutzer während seiner Nutzung und zieht daraus parallel zu den ablaufenden normalen Adaptionen Schlüsse über Veränderungen der Kontextadaption. Zum Beispiel könnte die häufigste Adaption für diesen Benutzer zur Standardeinstellung erhoben oder eine niemals verwendete Adaption entfernt werden.

Die Unterscheidung in aktiv und passiv macht wegen der geringen Unterschiede im Modellierungsergebnis vor allem als Grundlage für Entscheidungsprozesse während der Modellierung einer konkreten Benutzerschnitt-

stelle einen Sinn. Technisch gesehen werden beide Möglichkeiten als eine Form Kontextadaption ausgedrückt und auch umgesetzt.

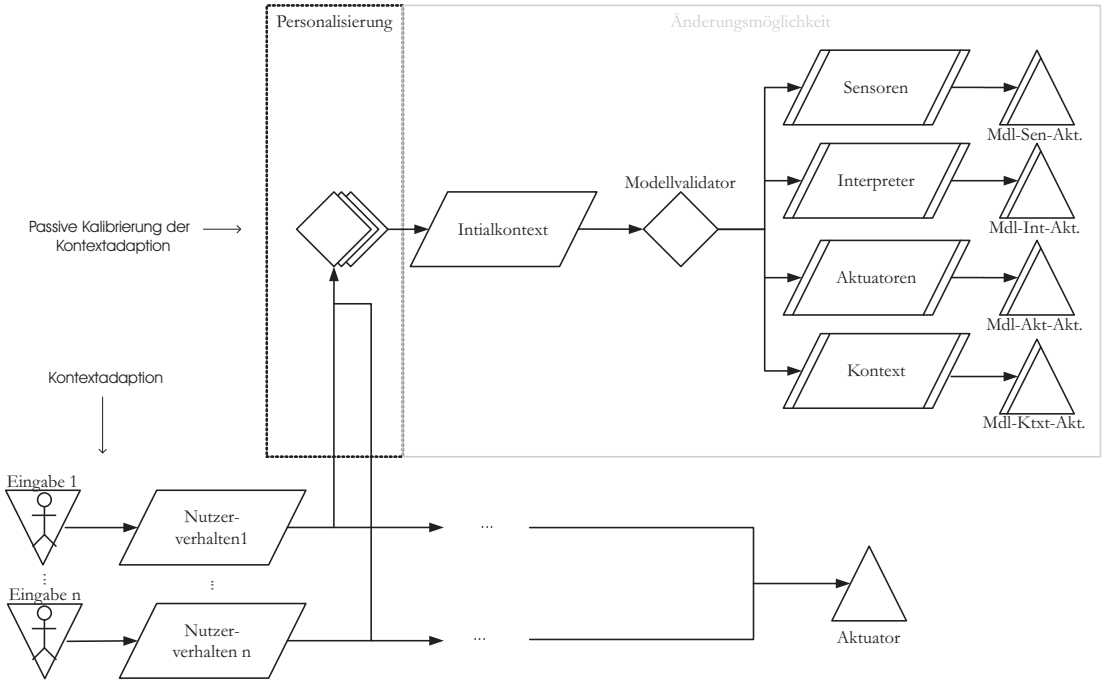


Abb. 60: Prinzip passiver Kalibrierung

SUB Monitoring

Neben Ein- und Ausgabe von Kontextadaptionsbeschreibungen kann die Kalibrierung wie eingangs erwähnt auch noch aus Interaktionen bestehen, die der Nachverfolgung oder Vorauserkennung von SUB-Phänomenen dienen. Der Prozess ist vergleichbar dem Debugging von Anwendungen, mit dem Unterschied, dass die Kontextadaption und SUB im Fokus stehen. SUB Monitoring muss zum Zeitpunkt des Entwurfes der ubiquitären Anwendung bereits berücksichtigt und die Kontextadaption dementsprechend instrumentiert werden. Das Framework aus Kapitel 6 hält dafür einige Unterstützung bereit.

SUB Marker erlauben es dem Benutzer, zur Laufzeit aufgetretene SUB-Phänomene einfach und zeitnah zu markieren und dann zu einem späteren Zeitpunkt in einer vollständigen Kalibrierung aufzugreifen. Dies ist notwendig, da eine vollständige Kalibrierung oft nicht ubiquitär ist (siehe aktive Ein- und benutzerinitiierte Ausgabe weiter oben). Sie wird daher nach dem Auftreten eines SUB meist so lange verzögert, bis eine Situation erreicht wird, in der eine direkte und exklusive Interaktion mit dem Benutzer möglich ist. In der

Zwischenzeit kann sich das System aber durch weitere Adaptionen so weit verändert haben, dass der ursprüngliche Grund für das beanstandete SUB nur noch schwer nachzuvollziehen ist.

Die Markierung von SUB basiert auf der Tatsache, dass Kontextelemente eine Historie besitzen. Zusammen mit der Tatsache, dass die Kontextadaptionen selbst in Form ihrer Beschreibung im Kontext (Metamodell) gespeichert wird, erlaubt dies bis zu einem gewissen Grad, den kompletten Zustand einer Kontextadaption zu sichern und wiederherzustellen. SUB Marker können durch spezielle Kalibrierungsadaptionen realisiert werden. Diese werden von einem Mediator-Sensor (beispielsweise ein Knopf auf einer Fernbedienung) aktiviert und speichern das Auftreten eines SUBs und dessen Zeitindex in einem Kontextelement. Mithilfe dieses Zeitindexes kann der gespeicherte Zustand zu einem späteren Zeitpunkt genauer untersucht werden.

Abb. 61 zeigt ein Beispiel der Verwendung von SUB Markern in der Smarthome Anwendung der Fallstudie aus Kapitel 2. Über einen Filterinterpreter wird der Druck einer bestimmten Taste einer Fernbedienung aus der normalen Fernbedienungsverarbeitung herausgefiltert und als zeitnahe Unzufriedenheit des Benutzers mit einem bestimmten Adaptionsverhalten interpretiert. Als Reaktion darauf wird zunächst ein Zeitstempel gesichert, mit dessen Hilfe sich später aus der Kontexthistorie der gegenwärtige oder vorangegangene Systemzustand (der Kontextadaption) rekonstruieren lässt, beispielsweise, um ihn später in eine Simulationsumgebung zu laden. Darüber hinaus wird durch ein Logging Subsystem in Form eines Aktuators ein Logbucheintrag erzeugt, der die kürzlich veränderten Kontextelemente und die davon getriggerten Sensor-, Interpreter- und Aktuatorelemente namentlich auflistet.

Wie im Beispiel gezeigt, können SUB Marker dazu verwendet werden, die zum Zeitpunkt des SUB herrschenden Zustände des Kontextadaptions-Subsystems einer ubiquitären Anwendung in eine Simulationsumgebung zu übertragen.

SUB Simulationen erlauben es generell, bestimmte Umgebungssituationen künstlich zu erzeugen und so das Entstehen eines SUB zu verfolgen, das Auftreten eines für die Zukunft vermuteten SUB zu bestätigen, oder eine Kalibrierungsmaßnahme auf ihre Wirksamkeit hin zu überprüfen. Dabei darf natürlich die Ubiquitätsbedingung der Anwendung nicht verletzt werden. Das Framework stellt daher eine Unterstützung bereit, die Kontextadaption einer laufenden Anwendung samt dem Kontext zu klonen und in eine Simulationsumgebung zu übertragen. Das Klonen erfolgt durch einen speziellen Aktua-

tor, der das gesamte Metamodell der Kontextadaption aus dem Kontext einer Anwendung ausliest und über einen speziellen Sensor in den Kontext einer zweiten Anwendungsinstanz überspielt. Natürlich können die an die verschiedenen Dienste der vervielfältigten Kontextadaption gebundenen Komponenten nicht ebenso leicht dupliziert werden, zumal es sich ja um Komponenten handeln könnte, die von physikalisch begrenzten Ressourcen (z.B. Geräten) abhängen.

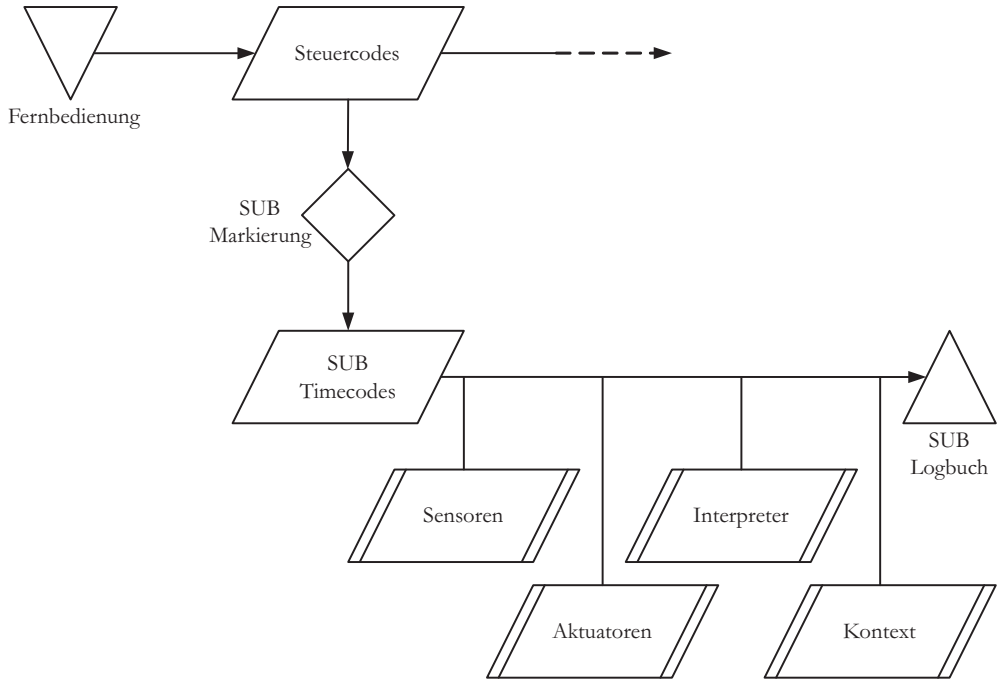


Abb. 61: Beispiel für die Verwendung von SUB Markern in der Fallstudie

Bei der Überspielung in die Simulation werden daher alle Sensoren und Aktuatoren an spezielle Simulationskomponenten gebunden. Der Simulationssensor erlaubt das Abspielen einer Kontexthistorie oder die Manipulation einzelner Kontextwerte. Der Simulationsaktuator erzeugt dagegen beispielsweise einheitliche Ausgaben aktivierter Aktuatoren (z.B. als Logbuch). SUB Marker und Simulationen zusammen erlauben somit ein genaueres Eingrenzen eines aufgetretenen SUB-Phänomens.

7.3.3 Techniken für Ausschnittsbildung und Navigation

Die größten Schwierigkeiten bei der Umsetzung der im letzten Abschnitt dargelegten Entwurfsprinzipien der Kontrolllogik (vergleichbar mit *control* des MVC) für eine Benutzerschnittstelle der Kalibrierung ist eine benutzerfreundliche Darstellung (*view*) des zugrunde liegenden Modells einer Kontextadaption. Das in Kapitel 5 in Form von K-Modellen definierte Modell einer

Kontextadaption war zwar in Anlehnung an ein intuitives Verständnis der Kontextadaption konstruiert worden, bleibt aber nach wie vor ein allgemeines technisches Modell. Der spezifische Einzelfall einer Kalibrierungsmöglichkeit bezieht sich dagegen in der Regel nur auf einen Ausschnitt oder eine Untermenge der darin enthaltenen Details. Daraus ergibt sich ein noch ungenutztes Potenzial für Vereinfachungen.

Angenommen eine Kalibrierungsmöglichkeit beschränkt sich auf die Auswahl zwischen mehreren alternativen Quellen einer Sensorinformation, etwa für eine Wettervorhersage. Damit ist in diesem Fall lediglich ein Teil des K-Modells für den Benutzer überhaupt relevant, also in der Regel der eine Sensor, höchstens aber alle Elemente, die mit ihm in Verbindung stehen. Zudem sind von diesem Sensor auch nicht alle Details seiner Spezifikation von Interesse. Von Belang ist lediglich eine Darstellung des Identifikators der an den Sensordienst gebundenen Komponenteninstanz, beispielsweise in Form einer URL. In Bezug auf die Visualisierung des Modells in der Benutzerschnittstelle dieser spezifischen Kalibrierung kann diese zum Beispiel zu einer einfachen Auswahlliste oder einem Textfeld vereinfacht werden.

Auf die Wirksamkeit der Kalibrierungsmaßnahmen haben solche Vereinfachungen im Rahmen einer einzigen Kalibrierungsmöglichkeit nicht notwendigerweise einen mindernden Einfluss, da eine ubiquitäre Anwendung zugleich mehrere spezialisierte Kalibrierungsmöglichkeiten besitzen und auch parallel einsetzen kann. Entscheidend ist vielmehr die Summe aller möglichen Kalibrierungsmaßnahmen.

Zum Beispiel besitzt die in dieser Arbeit betrachtete Fallstudie eine Reihe von spezialisierten aber stark eingeschränkten Kalibrierungsmöglichkeiten. Daneben gibt es aber auch noch eine Möglichkeit, über eine Reinitialisierung des in 6.1.5 beschriebenen Bootvorganges im laufenden Betrieb das bisherige Modell der Kontextadaption durch ein komplett neues initiales Kontextmodell (z.B. in Form einer auf einem Hardwarebaustein gespeicherten Datei oder einem Software-Download) zu überschreiben. Auf diese Weise ist in jedem Fall eine vollständige Kalibrierbarkeit gegeben.

Im Rahmen des Entwurfes der darstellenden Anteile (*view*) einer Benutzerschnittstelle für eine konkrete Kalibrierungsmaßnahme unterstützt die in diesem und dem nächsten Unterabschnitt beschriebene Methodik zusammen mit dem Framework aus Kapitel 6 zwei prinzipielle Vereinfachungsgrundansätze:

Zwei prinzipielle Vereinfachungsgrundsätze

- Die Komplexität der Kontextadaption einer ubiquitären Anwendung kann durch Bildung geeigneter Ausschnitte reduziert werden. So erhält man beispielsweise eine übersichtliche und vereinfachte Darstellung, um sich zu orientieren.
- Die Beschreibung der Kontextadaption aus 6.3 kann auf verschiedene Weise syntaktisch transformiert oder unter Vernachlässigung von Details projiziert werden, um eine intuitiv verständliche Darstellung zu erreichen. So lassen sich statt der grafischen Symbole beispielsweise auch Hardware-Bedienelemente, Sprachausgabe oder Gestenerkennung als Eingabe vorstellen, um einen Teil der Beschreibung einer Kontextadaption zu präsentieren oder zu verändern.

Die Möglichkeiten dazu sind sehr vielfältig und teilweise auch sehr stark abhängig von der jeweiligen Anwendungs- und damit der Nutzerdomäne. Der folgende Abschnitt kann daher nur sehr allgemeine methodische Hinweise für die Bildung von Ausschnitten geben. Gleiches gilt für die verschiedenen denkbaren Darstellungsformen und Detailreduzierungen der allgemeinen Beschreibung von Kontextadaptionen aus 6.3, die im darauf folgenden Abschnitt 7.3.4 näher behandelt wird.

Grundprinzip der Ausschnittsbildung

Entkoppelungseigenschaften erleichtern die Ausschnittsbildung.

Aufgrund der Entkoppelungseigenschaft der K-Modelle, die eine Trennung in Teilprozesse und Elemente erlaubt, fällt es sehr leicht, die Beschreibung einer Kontextadaption entsprechend in sinnvolle Teilausschnitte zu zerlegen. Da darüber hinaus elementübergreifende Konsistenzeigenschaften nur in Form von weiteren K-Modell-Elementen (z.B. Sensorenredundanz in 7.2.2) ausgedrückt werden dürfen, ist sichergestellt, dass diese Eigenschaften bei der Ausschnittsbildung berücksichtigt bleiben. Für die Partitionierung selbst kommen die folgenden Techniken in Frage.

Faltung

Diese Partitionierungstechnik basiert auf der grundlegenden Entkoppelung einer Kontextadaption in die drei Schritte der Kontextgewinnung, Adaptionsentscheidung und Adaptionsdurchführung (siehe auch Modellbildung 5.1). Die Kommunikation zwischen datenverarbeitenden Elementen wird dabei indirekt über die Kontextelemente abgewickelt. Eine semantisch korrekte und konsistente Betrachtung auch lediglich eines Partners einer teilprozessübergreifenden Kommunikationsbeziehung ist damit ohne weiteres möglich. Dies erlaubt es, die Darstellung eines K-Modells entlang einer solchen Kontextgrenze zu *falten* und einzelne Teilabschnitte zu verbergen, also sozusagen

umzuklappen. Ein so umgeklappter Adaptionsschritt entspricht dann einer Art von Blackbox-Sicht, die lediglich ihr äußeres Interface in Form der an den Teilprozessgrenzen befindlichen Kontextelemente enthüllt, die innere Struktur aber verbirgt. Aufgrund der Eigenschaften der Kontext-Elemente lässt sich eine solche Faltung nicht nur an den Teilprozessgrenzen der Kontextadaption durchführen, sondern auch an jeder anderen Einteilung eines K-Modells, sofern für diese eine Grenze in Form von Kontextelementen definiert werden kann.

Die durch Faltung erzeugten Ausschnitte eines K-Modells eignen sich besonders zur Reduktion des Modells auf eine Liste, die als ein Verzeichnis, Index oder ein anderes Mittel zur übersichtlichen Navigation dienen kann.

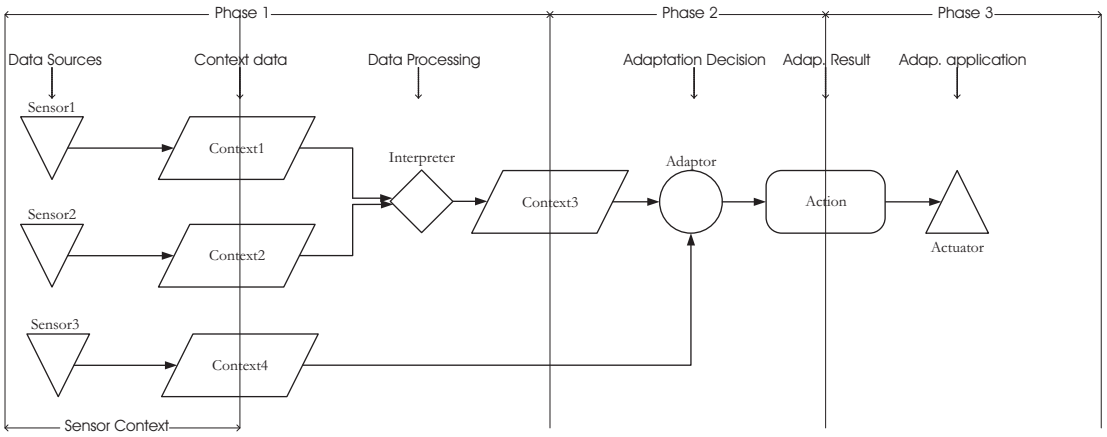


Abb. 62: Segmentierung an Teilprozessgrenzen

Für die Durchführung einer Faltung wird die Darstellung des K-Modells im ersten Schritt so angeordnet, dass sich die Kontextelemente an den Prozess- oder anderen Grenzen ausrichten lassen (siehe Abb. 62). Elemente höherer Ordnung, die auf Kontextelementen basieren, wie beispielsweise die Aktionen, können für die Faltung wie ein einfaches Kontextelement behandelt werden. Außerdem sollte die Darstellung eine Richtungsorientierung bezüglich des Datenflusses von Kontextinformationen von Eingabe (Sensoren) zur Ausgabe (Aktuatoren) besitzen. In Abb. 62 wurde eine Orientierung von links nach rechts gewählt.

Im nächsten Schritt können ein oder mehrere der so entstandenen Segmente verdeckt werden (siehe Abb. 63). Die Faltung selbst sollte in der Ausschnittsdarstellung deutlich markiert werden (in Abb. 63 durch eine angedeutete Falzlücke), damit es nicht zu Missinterpretationen kommen kann.

Durch Faltung entstandene Ausschnitte bleiben aufgrund der Orientierung und der Elemententkoppelung trotzdem sinnvoll interpretierbar. In Abb. 63

also beispielsweise durch die Reduzierung auf die Kontextein- und Ausgabemöglichkeiten des betrachteten Systems.

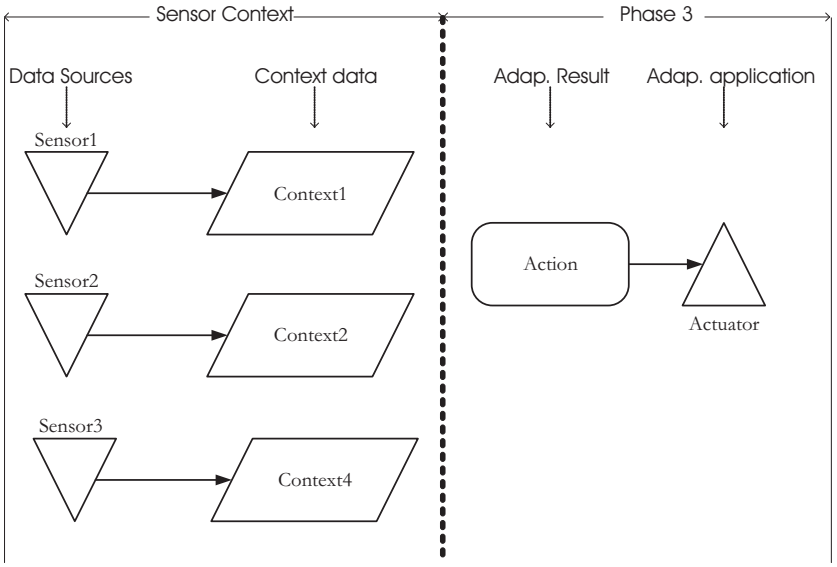


Abb. 63: Sensor-Aktion Faltung

Faserung

Die Partitionierungstechnik der Auffaserung macht sich die Eigenschaft von K-Modellen zu nutze, parallele und voneinander unabhängige Adaptionsstränge enthalten zu können. Vergleichbar der Faltung kann die Trennung dieser Adaptionsstränge (Threads) durch entsprechende Anordnung der Elemente einer K-Modell-Beschreibungstechnik zunächst sichtbar gemacht werden. Nach erfolgter Segmentierung können dann durch Herausnehmen oder Verstecken einzelner Abschnitte Ausschnitte der Adaption erzeugt werden.

Im Prototyp der Fallstudie aus Kapitel 2 wurde im Entwurf der Kalibrierungskonsole ebenfalls auf die Technik der Faserung zurückgegriffen, um das Gesamtmodell der Adaption in übersichtliche Teile zu partitionieren. Abb. 64 zeigt exemplarisch, wie ein größeres Modell der Adaption durch Teilung der Adaptionsfasern in zwei Ausschnitte zerlegt werden kann. Die grafische Darstellung (unten im Bild) zeigt den oberen Teilausschnitt des Modells (oben) nachdem zusätzliche syntaktische und optische Transformationen durchgeführt wurden. Auf diese wird in 7.3.4 noch einmal näher eingegangen.

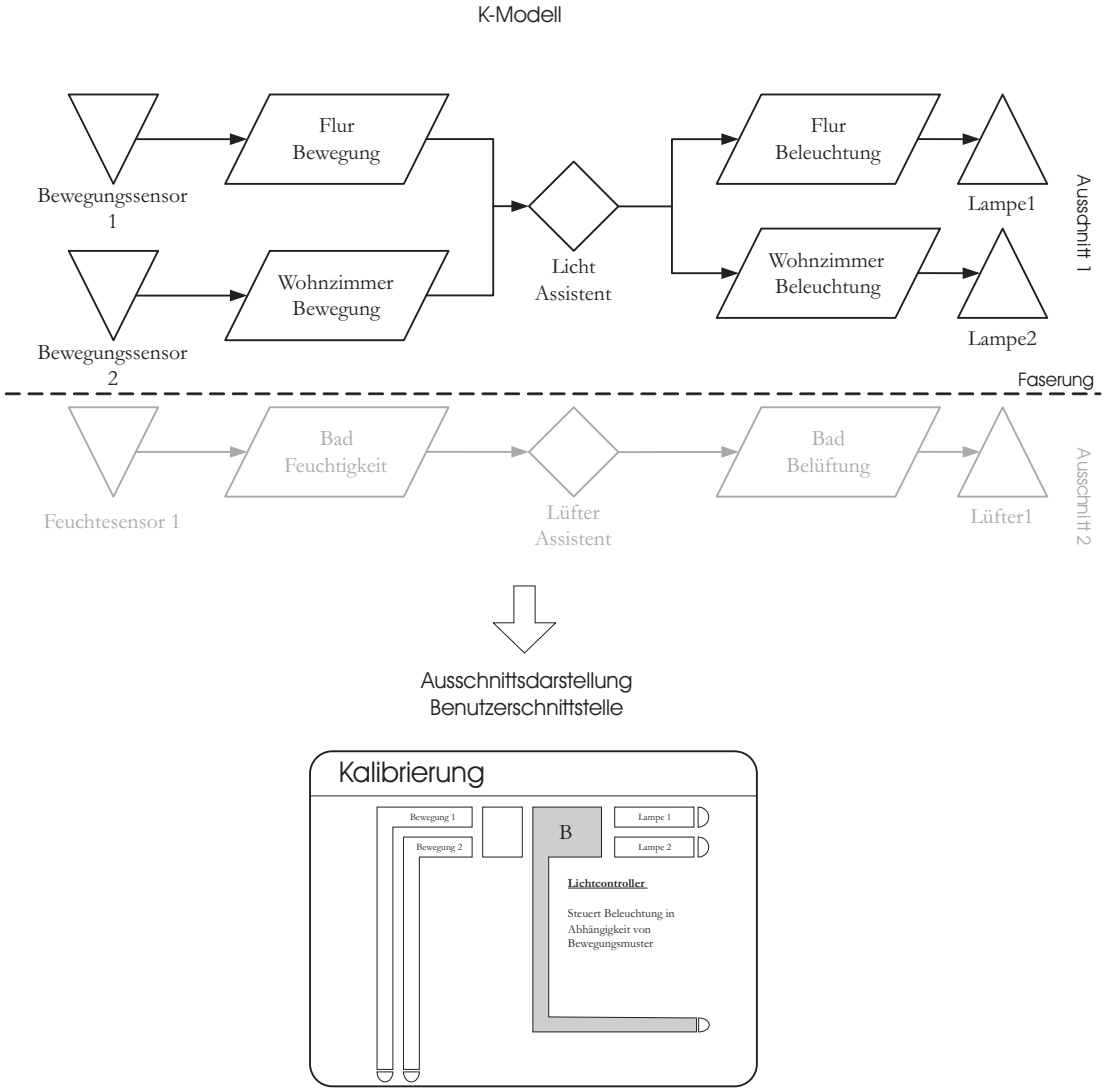


Abb. 64: Ausschnittsbildung durch Faserung am Beispiel der Fallstudie.

Aufspaltung

Partitionierungstechniken, wie die gerade beschriebene Faserung haben den Nachteil, dass sie sich lediglich auf voneinander unabhängige Elementgruppen der Beschreibung einer Kontextadaption anwenden lassen.

Um eine weitergehende Unterteilung zu erreichen, muss eine Möglichkeit geschaffen werden, bestimmte Elemente semantisch korrekt zwischen mehre-

ren Segmenten aufzuteilen, oder als Kopien mehreren Segmenten gleichzeitig zuordnen zu können.

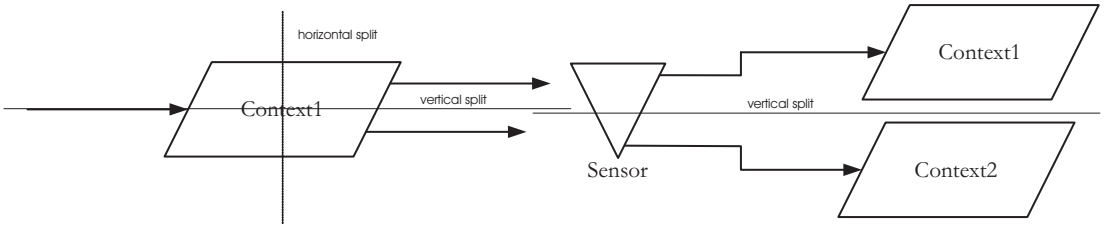


Abb 65: Horizontale und vertikale Kontextspaltung (l), Sensorspaltung (r)

Ohne semantische Einschränkung lassen sich lediglich Kontextelemente auf diese Weise aufspalten, da diese für die Erfüllung der Entkoppelungseigenschaft von K-Modellen bereits entsprechend konstruiert sind. Die Spaltung kann dabei entweder zwischen allen Ein- und Ausgaben erfolgen (vertikale Trennung) oder auch zwischen mehrfachen Ausgaben (horizontal). Eine Trennung zwischen mehreren Eingaben ist nicht möglich und auch nicht notwendig, da die Konstruktionsregeln für K-Modelle einen solchen Fall aus Konsistenzgründen verbieten (siehe auch Abb. 68).

Ebenfalls vertikal lassen sich die Sensorelemente einer Beschreibung von Kontextadaption spalten. Dies ist fast ohne zusätzliche Einschränkungen möglich, da immer nur eine Kontextinformation gleichzeitig in alle angeschlossenen Kontextelemente geschrieben wird. Diese implizite Synchronität könnte zwar bei einer vertikalen Teilung verloren gehen, bleibt aber auch ohne Teilung aufgrund der Entkoppelungseigenschaften des K-Modells höchstens bis zum nächsten Kontextelement erhalten und müsste ab dort bei Bedarf explizit über die Historiefunktion wieder hergestellt werden.

Interpreter lassen sich auf die gleiche Weise spalten wie Sensoren, sofern ihr Eingangskontext lediglich aus einem einzigen Element besteht, respektive alle Eingabekanäle für jeden Ausgabekontext dupliziert werden (siehe Abb. 66). Semantisch ist so eine Duplikation ohne weiteres möglich, da es sich bei einem Interpreter lediglich um einen Dienst handelt und in Kapitel 5 definiert worden war, dass ein und dieselbe Komponenteninstanz durchaus mehrere Dienste gleichzeitig erfüllen kann. Wie im Falle des Sensors kann auch keine Synchronität verloren gehen, da diese ohnehin durch das nächste Kontextelement entkoppelt würde.

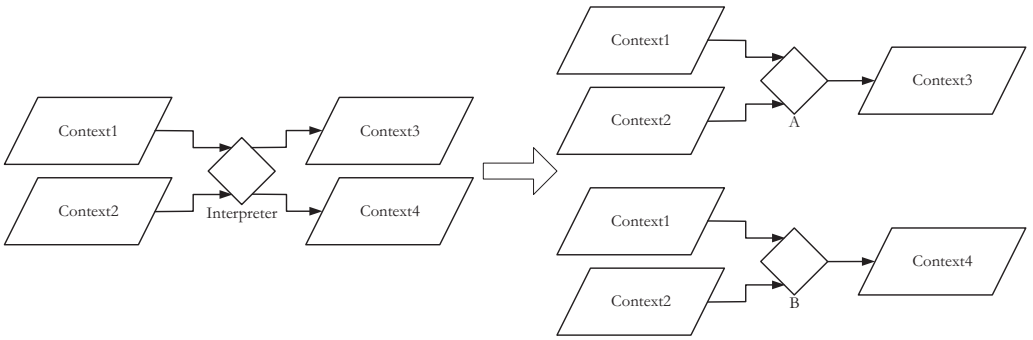


Abb. 66: Interpreterspaltung

Eine Spaltung mithilfe der Eingabekontexte wäre dagegen nur auf Basis von Spaltbedingungen möglich, die sich auf die Komponentenbindung des Interpreterelementes beziehen. Ein Beispiel dafür wäre die Existenz einer Zerlegung in mehrere voneinander unabhängige Ein-/Ausgabebeziehungen. Da in K-Modellen aber unterschiedliche Beschreibungen für Komponentenbindungen zum Einsatz kommen können, lassen sich hier keine generell gültigen Spaltbedingungen angeben. Eine Ausnahme hiervon ist die Existenz einer Hierarchisierung oder Komposition auf der Ebene der Kontextadaptionsbeschreibung. Dies bedeutet, ein Interpreter lässt sich bezüglich seiner Eingänge spalten, wenn er selbst aus einer Komposition mehrerer Interpreter hervorgegangen ist. Die erlaubte Spaltung entspricht dann genau der Umkehrung der Komposition. Gleiches gilt für Aktuatoren, bei denen die Spaltung als Dekomposition einer vorangegangenen Hierarchisierung die einzige Teilungsmöglichkeit darstellt. (Abb. 67)

Hierarchisierung

Hierarchisierung oder Komposition von Elementen einer Beschreibung von Kontextadaption stellt die Umkehrung der Spaltung dar. Durch diese Art der Zusammenfassung mehrerer Elemente zu einem einzigen kann eine Detailabstufung der unterschiedlichen Darstellungen eines K-Modells und damit eine bessere Übersicht erzielt werden.

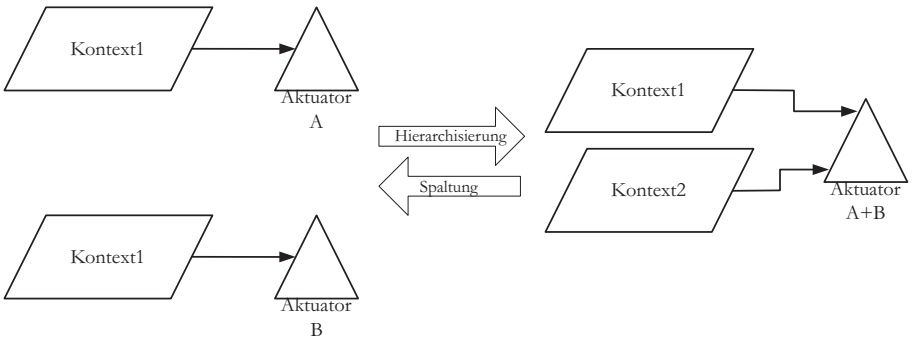


Abb. 67: Aktuatorspaltung als Umkehrung der Hierarchisierung

Semantisch wird die Komposition wiederum durch die Entkoppelung über Kontextelemente möglich. Als Folge davon lassen sich Sensoren, Interpreter und Aktuatoren immer durch einfache Addition ihrer Ein- und Ausgabekontexte kombinieren (vertikale Komposition). Sind die Spaltungsbedingungen erfüllt, lassen sich Kontextelemente vertikal kombinieren, beispielsweise als Umkehrung der in Abb. 66 gezeigten Spaltung. Eine generelle vertikale Komposition von Kontextelementen ist dagegen nicht möglich (Abb. 68). Dies wird aber bereits durch die Ein-Eingangsregel für Kontextelemente sichergestellt.

Serialisierung

In manchen Fällen kann es notwendig sein, aus Platz-, Übersichts- oder didaktischen Gründen die Beschreibung einer Kontextadaption systematisch in eine geordnete Folge von Ausschnitten zu serialisieren. Beispiele hierfür sind Handbücher und Referenzen, aber auch die Erzeugung von Indizes oder Verzeichnissen für die Navigation zwischen Ausschnitten.

Jede Serialisierung einer Kontextadaptionsbeschreibung basiert auf der Bildung einer Menge von Ausschnitten, deren Komposition das ursprünglich beschriebene Gesamtmodell ergibt sowie einer darauf definierten Ordnung, die das Auffinden eines bestimmten Ausschnittes erlaubt. Um die Definition einer solchen Ordnung zu erleichtern und eine vollständige Überdeckung zu gewährleisten, sollten die Ausschnitte mit den zuvor beschriebenen Techniken so gebildet werden, dass sie eine für die Ordnung geeignete Normalform

erfüllen. Im Folgenden werden vier Beispiele für solche Normalformen beschrieben.

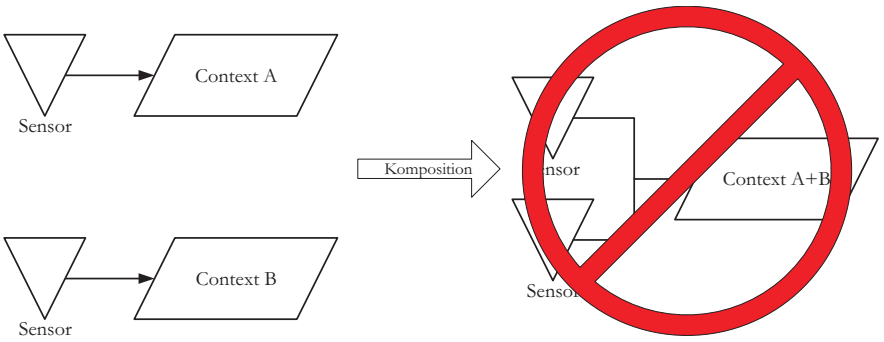


Abb. 68: Verbotene Kontextkomposition

- *Thread-Normalform*. Diese Normalform basiert auf der zuvor beschriebenen Partitionierungstechnik der vertikalen Faltung. Die Thread-Normalform besagt, dass zwischen den Ausschnitten keine Verbindungen durch gemeinsame Kontextelemente bestehen.
- *Elementarnormalform*. In der Elementarnormalform enthält jeder Ausschnitt genau ein Grundelement eines bestimmten Typs (Kontext, Sensor, Interpreter oder Aktuator) sowie alle seine Vorgänger- und Nachfolgerelemente.
- *Kategorienormalform*. Diese entspricht der Elementarnormalform, allerdings sind lediglich die Grundelemente eines Typs enthalten, die mit der gleichen Kategorie markiert sind. Die Kategorienormalform wird zum Beispiel verwendet, um lediglich die Interpreter zu serialisieren, die als Situation markiert wurden.
- *Gruppennormalform*. Im Framework Kapitel 6 wurde die Möglichkeit genannt, mehrere Grundelemente zu einer Gruppe zusammenzufassen. Gruppen sind ein virtueller Elementtyp, dem wieder eine Kategorie zugeordnet werden kann. Auf diese Weise werden höhere Elemente aus den Grundelementen konstruiert wie etwa die Aktionen.

Natürlich sind noch weitere Normalformen für die Serialisierung denkbar. Die häufigsten Serialisierungen basieren aber auf einer Liste der Situationen (Kategorienormalform) oder verschiedenen Formen der Gruppennormalform wie beispielsweise einer Liste der Aktionen. Serialisierungen können auch kombiniert werden, beispielsweise in Form einer Liste von Aktionen, die einer bestimmten Situation zugeordnet sind.

Die bereits aus Abb. 55 bekannte Kalibrierungsschnittstelle in Form einer Online-Hilfe für den Prototyp der ubiquitären Smarthome Anwendung aus der Fallstudie in Kapitel 2 benutzt für die Navigation zwischen den einzelnen Ausschnitten des Adaptionsmodells im Ganzen drei Serialisierungen (Abb. 69).

Die erste indiziert jeweils alle Interpreter, die zur Kategorie der Situationen gehören. Die zweite basiert auf Interpretern, die als Assistenten markiert wurden. In der Domäne der Fallstudie kontrolliert ein Assistent mehrere Geräte und Funktionen in verschiedenen Situationen. Geräte sind schließlich domänenspezifische Gruppen von Sensoren und Aktuatoren, die einem physikalischen Gegenstand zugeordnet werden können. Mithilfe der sich daraus ergebenden Einteilung des Adaptionsmodells kann auf einfache Weise in der Benutzerschnittstelle zwischen den Modellausschnitten navigiert werden.

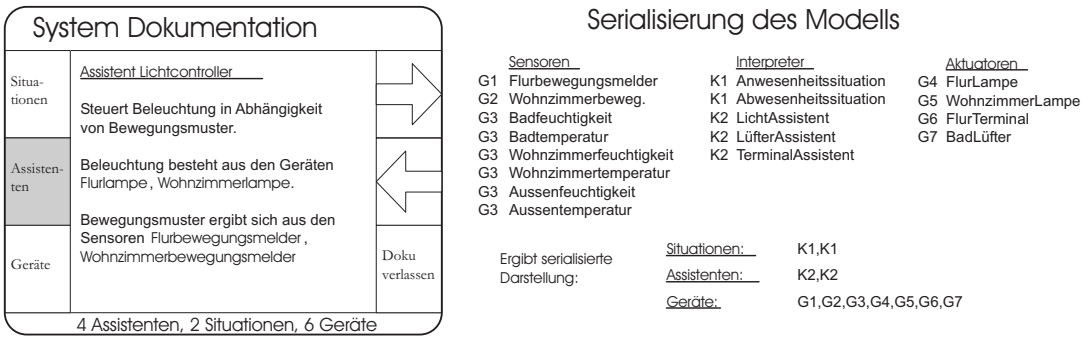


Abb. 69: Serialisierungsschemata der Fallstudie

7.3.4 Einbindung multimodaler Interaktionsformen

Zentrales Element der Benutzerschnittstelle für Kalibrierung ist neben dem richtigen Inhaltsausschnitt auch die Form der direkten und bewussten Interaktion zwischen System und Benutzer. Kernproblem dabei sind aber geeignete Repräsentationen der Beschreibungen von Kontextadaption, die für die Kalibrierung benötigt werden. Syntaktisch hängt die Eignung der für Kalibrierung verwendeten Notation jedoch stark von der jeweiligen Zielgruppe und dem Verwendungszweck ab. Die in 6.4 definierte Spezifikationstechnik war bereits unter dem Gesichtspunkt allgemeiner Eignung für Endbenutzer konstruiert worden. Die Gemeinsamkeit der Zielgruppe besteht darin, dass die Verwendung klassischer Spezifikationstechniken wie UML wegen eines

zu hohen Lernaufwandes und zu hoher Komplexität nicht sinnvoll ist. Statt dessen wird versucht, die Spezifikationsmöglichkeiten des Benutzers durch eine Menge vorgegebener oder einzeln zu konfigurierender Bausteine einzuschränken, deren abstrakte Bedeutung für die Kontextadaption und deren Kombinierbarkeit durch eine von vier Rollen (Kontextelement, Sensor, Interpreter, Aktuator) vorgegeben ist.

Für bestimme Anwendungszwecke kann aber sogar das zu kompliziert sein. Für den Entwurf der Benutzerschnittstelle der Kalibrierung eines konkreten Systems muss diese Grundbeschreibungstechnik daher noch in eine jeweils geeignete Form transformiert werden. Daraus ergeben sich die folgenden, im Entwurf einer konkreten Benutzerschnittstelle für Kalibrierung zu bearbeitenden Fragestellungen:

- Besteht die Interaktivität aus mehreren unidirektionalen oder einer bidirektionalen Schnittstelle?
- Welche optischen (medialen) Transformationen gelten für die Form der Beschreibungstechnik in jeder dieser Schnittstellen?
- Welche syntaktischen Transformationen gibt es?

Entwurf der Interaktivitätsstruktur

Zur Beantwortung der ersten Fragestellung muss man sich zunächst darüber klar werden, dass gerade in ubiquitären Anwendungen Interaktivität nicht zwangsläufig bidirektional in einer einzigen Form (*Mode*) stattfinden muss. Für die Kalibrierung bedeutet dies, dass die Unterrichtung des Benutzers über den momentanen Zustand der Kontextadaption nicht in der gleichen Form stattfinden muss, in der er seine Änderungswünsche formuliert oder formuliert hat. Im besonderen Maß gilt das für die passive Kalibrierung (siehe Abb. 70C). Besteht dagegen eine aktive und bewusste Interaktionsbeziehung, ist die Benutzerschnittstelle in beiden Richtungen in der Regel zumindest ähnlich (Abb. 70A), kann aber über mehrere Medien hinweg verteilt sein (multimodal Abb. 70B,C). Zum Beispiel könnten sprachgesteuerte und visuelle Formen kombiniert eingesetzt werden.

Eine aktive Eingabe der Kalibrierung wird oft, aber nicht zwingend mit einer benutzerinitiierten Ausgabe kombiniert. Das bedeutet, dass die Interaktionsstruktur üblicherweise zunächst damit beginnt, den Benutzer auf dessen eigenen Wunsch hin über den gegenwärtigen Zustand der Kontextadaption zu informieren (Abb. 70B). Der Benutzer entscheidet daraufhin anhand des dargestellten Systemzustandes, ob eine Kalibrierung notwendig ist. Diese wird dann in einer Folge von Änderungs- und optionalen Rückkopplungsschritten (Warnungen, Fehlermeldungen etc.) durchgeführt. Die Änderung

am Ende noch einmal komplett zu dokumentieren ist dagegen in der Regel nicht notwendig, da der Benutzer die gewünschten Modifikationen ja selbst bereits formuliert hat und dadurch kennt.

Beispiele der Interaktionsstruktur einer Kalibrierung

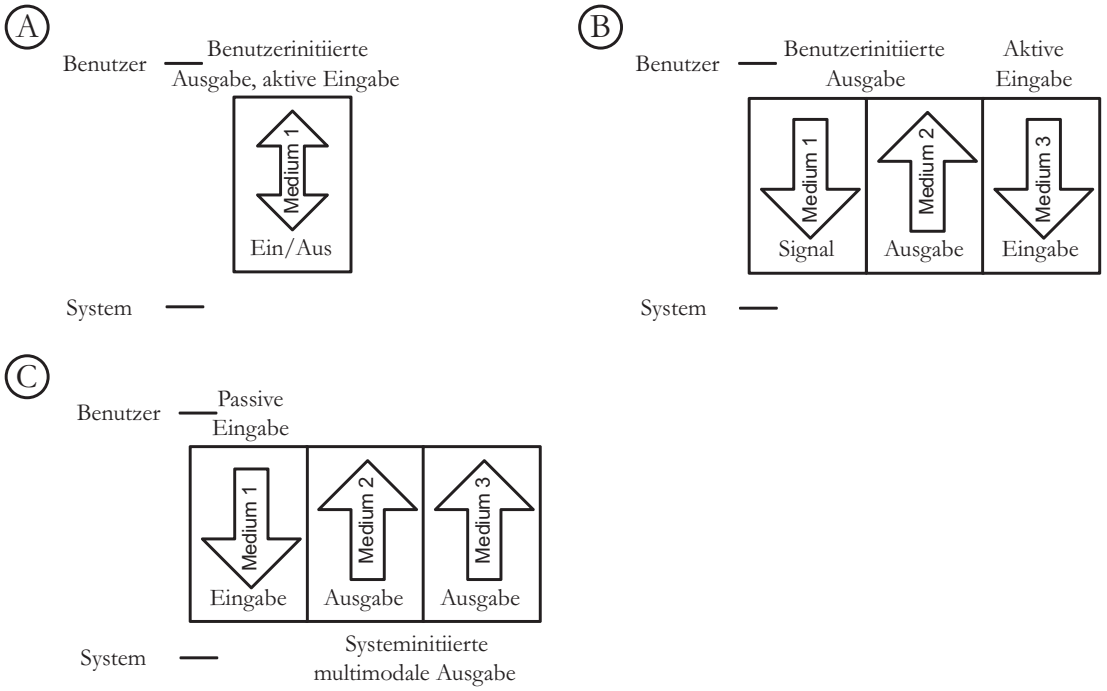


Abb. 70: Mögliche Interaktionsstrukturen der Kalibrierung

Die passive Eingabe der Kalibrierung wird dagegen meist mit der systeminitiierten Ausgabe kombiniert. Dabei wird dann das unbewusste Benutzerverhalten (z.B. erkennbare Unzufriedenheit) als Eingabe der Kalibrierung verwendet. Am Ende wird der Benutzer dann optional darüber informiert, dass und welche Änderung stattgefunden haben. Abseits der beiden häufigsten Interaktionsabfolgen der Kalibrierung sind aber auch andere Kombinationen denkbar, beispielsweise eine systeminitiierte Ausgabe, die den Benutzer zu einer aktiven Eingabe veranlasst, beispielsweise wenn das System sich durch Rückfragen oder Ankündigung vor der Durchführung einer möglicherweise gefährlichen Aktion beim Benutzer bemerkbar macht und ihm damit die Chance zur Intervention gibt.

Der Prototyp der ubiquitären Smarthome Anwendung aus der Fallstudie in Kapitel 2 benutzt für die Kalibrierung das in Abb. 71 dargestellte Interaktionsmuster. Die Grundidee dabei war, dass der Benutzer möglichst aus eigenem Antrieb einen Verdacht für zukünftiges oder aufgetretenes SUB entwickelt. Damit dies auch für weniger offensichtliche SUB-Phänomene gelingt, wird er von verschiedenen systeminitiierten bewussten und unbewussten Ausgabekalibrierungen unterstützt, wie beispielsweise der in Abb. 56 abgebildeten Heuristik für SUB kritische Situationen.

Im Anschluss verifiziert er seine Vermutung auf Wunsch über die Online Hilfe. Ist das SUB und sein Grund identifiziert, begibt er sich aus Sicherheitsgründen an die spezielle Kalibrierungskonsole aus Abb. 59 und führt dort die notwendigen Änderungen durch, indem er den entsprechenden Teil der Adaption zunächst über die Hardwarekonsole für eine Änderung freigibt, um dann über den Softwareeditor die notwendigen Änderungen durchzuführen.

Interaktionsstruktur der Kalibrierung des Prototypen

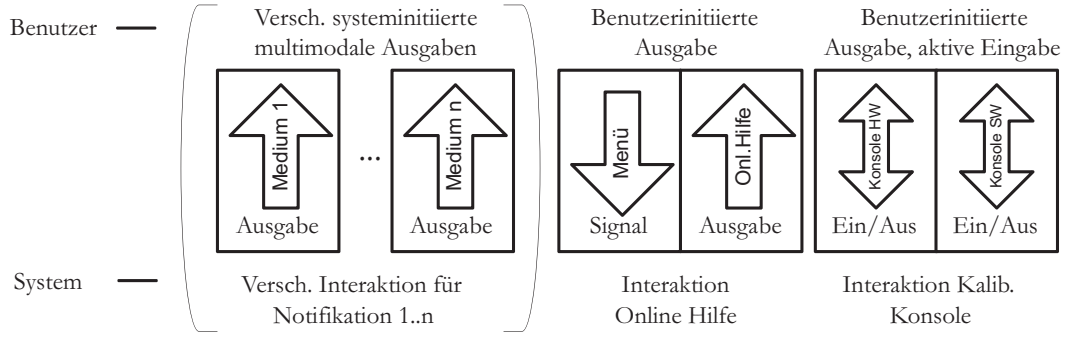


Abb. 71: Interaktionsstruktur der Kalibrierung des SmartHome Prototyp.

Die einzelnen Kalibrierungsschritte können dabei, müssen aber nicht für den Benutzer räumlich verteilt sein. Für das Beispiel eines nicht erreichbaren nichtmobilen Gerätes etwa, könnte diese potenziell SUB-kritische Situation dem Benutzer auch durch ein Blinken des entsprechenden Steckmoduls der Kalibrierungskonsole signalisiert werden.

Wird dies vom Benutzer direkt wahrgenommen, kann er sich natürlich auch direkt zu der Kalibrierungsstation begeben und auf dem dortigen Terminal die Onlinehilfe einsehen, bevor er durch ein Umstecken des Steckmoduls in den Programmierslot automatisch in den Kalibrierungseditor wechselt.

Skizziert man den Entwurf der Interaktionsstruktur für eine konkrete Benutzerschnittstelle der Kalibrierung auf die in Abb. 70 und Abb. 71 gezeigte Weise, kann man daraus bereits für den weiteren Entwurf wichtigen Informationen ablesen:

- Die Unterteilung in die einzelnen Teilschnittstellen *erkennen*, *verstehen* und *ändern*.
- Aus wie vielen uni- oder bidirektionalen Interaktionen eine solche Teilschnittstelle aufgebaut ist.
- Welche Modalitäten in jeder Einzelinteraktion involviert sind.

Für das Beispiel des Prototyps in Abb. 71 erkennt man, dass die Teilschnittstelle der Erkennungsphase aus verschiedenen unidirektionalen Notifikationen in verschiedenen Modalitäten aufgebaut ist (Blinken der Steckmodule, Töne beim Erkennen mobiler Geräte, Statusmeldungen etc.). Die Verstehen-Phase besteht aus der Benutzerinitiierung und der Ausgabe der Online Hilfe. Die Änderungsphase besteht dann letztendlich aus der multimodalen Kombination zweier bidirektionaler Interaktionen, nämlich dem Umstecken der Steckmodule und dem Bedienen eines Editors.

Mediale Transformationen

Trotz der sehr unterschiedlichen Beschaffenheit arbeiten alle im Entwurf der Interaktionsstruktur festgelegten Teilschnittstellen letztendlich auf dem gleichen Modell der Kontextadaption, nämlich dem für die jeweilige Anwendung gültigen und in deren Kontext als Spezifikationsbeschreibung gespeichertem K-Modell. Allerdings in vielen unterschiedlichen Darstellungsformen.

Nach dem Entwurf der Interaktionsstruktur für die Kalibrierungsschnittstelle müssen daher für jede Interaktionsform im letzten Schritt die medialen Transformationen der Beschreibungstechnik aus 6.4 festgelegt werden. Darunter versteht man Änderungen der Repräsentationsform der Elemente der Beschreibungstechnik aus 6.4. Dabei kann es sich um einfache grafische Veränderungen handeln, aber auch um Transformationen in völlig andere Medienformen, beispielsweise eine Übertragung in Sprache. Gerade solche Me-

dienwechsel finden sich in ubiquitären Systemen besonders häufig, da auch situationsbedingt die Verfügbarkeit eines optischen Wiedergabemediums nicht immer vorausgesetzt werden kann (Ubiquitätsbedingungen). Aber auch einfache Änderungen des optischen Designs können abseits technischer Notwendigkeiten eine psychologische, ergonomische oder wirtschaftliche Bedeutung haben, etwa für die Anpassung der Schnittstelle an das Produkt oder markenrechtliche Abgrenzungen, beispielsweise durch die Verwendung bestimmter Farbkombinationen.

Das Framework aus Kapitel 6 unterstützt die meisten medialen Transformationen ganz einfach durch eine standardisierte Bereitstellung einer Spezifikationsbeschreibung des gegenwärtigen K-Modells im Kontext einer Anwendung. Die Beschreibung kann in Form eines XML-Dokumentes ausgelesen werden und so leicht als Basis (Modell) für die medialen Umformungen einer konkreten Kalibrierungsschnittstelle dienen.

Darüber hinaus hält das Framework aber auch noch eine besondere Unterstützung für den Entwurf einer ganz speziellen medialen Transformation bereit, die einerseits direkt, andererseits auch als Zwischenschritt für weitere Transformationen fungieren kann.

Für bestimmte Anwendungszwecke kann es nämlich notwendig sein, die in der Beschreibungstechnik aus 6.4 ausgedrückten expliziten Adaptionsinformationen innerhalb einer anderen Spezifikation zu integrieren (maskieren). *Optische Maskierung* (OM) bedeutet dabei nicht, dass die Informationen verloren gehen. Die Informationen über die Kontextadaption tauchen lediglich implizit innerhalb einer anderen Beschreibungstechnik auf. Das Framework unterstützt die OM auf folgende Weise:

Aus den Beschreibungen der Elemente einer Kontextadaption gemäß 6.4, genauer gesagt aus den darin über die Dienstebindung referenzierten Komponentenimplementierungen, können informelle Beschreibungen der genauen Funktion extrahiert werden.

> Zum Beispiel kann die Komponente, die zur Laufzeit an einen in der Kontextadaptionsbeschreibung definierten Sensor gebunden wurde, eine kurze Beschreibung enthalten, etwa dass es sich um einen Feuchtigkeitssensor an einem bestimmten Ort oder mit einer bestimmten Genauigkeit handelt.

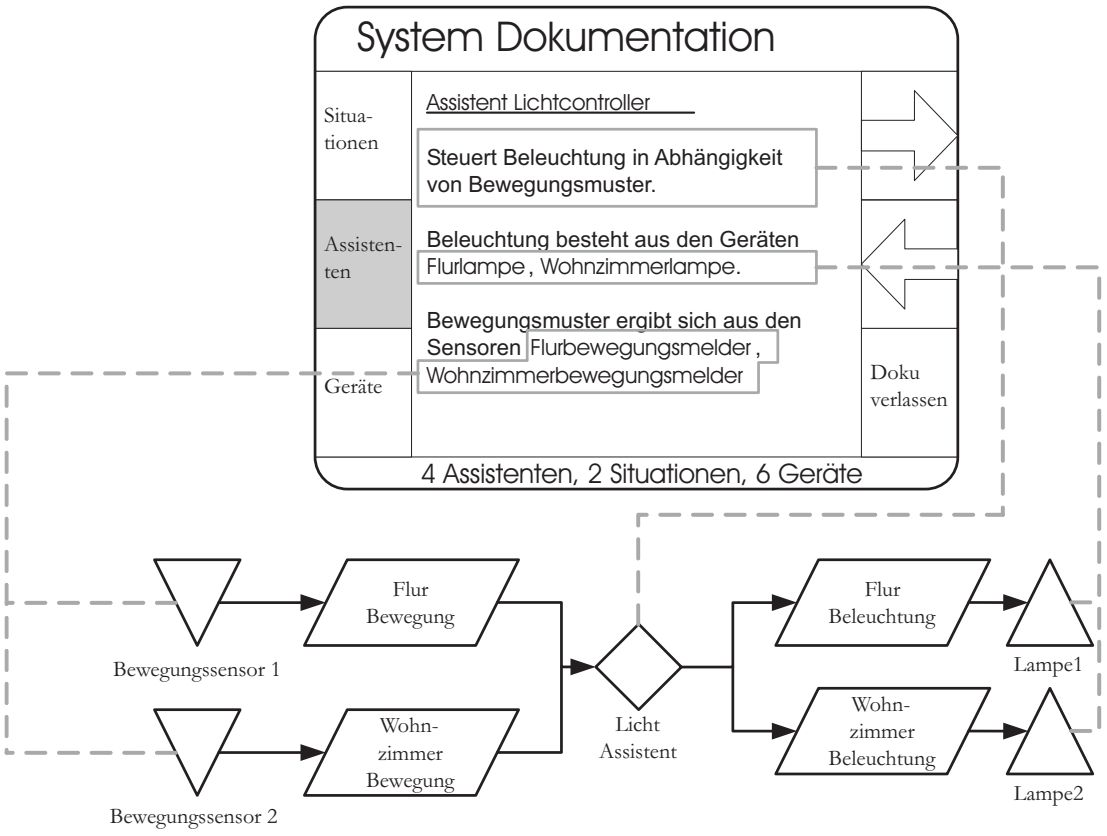
Das Framework unterstützt damit die OM einer Kontextadaptionsbeschreibung beispielsweise in Form eines Fließtext. Die Kontextadaption wird dabei in eine Kombination aus Textstruktur und Schlüsselsätze transformiert.

Eine einfache Adaption wie die Steuerung des Lüfters aus Kapitel 2 kann damit in folgender Beschreibung optisch maskiert werden:

Die Aktivierung der Aktion "*Lüfter (Bad) einschalten*", erfolgt, wenn *die Feuchtigkeit mehr als 60% beträgt. Die Feuchtigkeit* wird durch den Sensor "*Feuchtigkeit im Bad*" gemessen.

Die kursiven Teile sind jeweils die Textbausteine der gebundenen Komponenten. Die Adaptionsbeschreibung verbirgt sich in den restlichen Schlüsselwörtern, Teilsätzen und der Satzanordnung.

Abb. 72 zeigt zusätzlich ein anderes Beispiel für OM, wie es in der Online-Hilfe für den Prototyp der Fallstudie seine Verwendung findet.



K-Modell

Abb. 72: Verwendung von Optischer Maskierung für die Fallstudie

Die Verwendungsmöglichkeiten der so definierten optischen Maskierung sind vielfältig (Onlinehilfe, gedruckte Dokumentation, Logbücher etc.) und können darüber hinaus auch noch als Ausgangspunkt für weitere mediale Transformationen dienen, etwa für eine Interaktion, die auf einem Sprachdialogsystem basiert. Eine Sprachausgabe ist ein typisches Beispiel für die mediale Transformation einer K-Modell Beschreibung, die auf einen Zwischenschritt zurückgreift, der mithilfe der OM erzeugt wurde. In diesem Fall wird der erzeugte Fließtext des letzten Beispiels einfach vorgelesen.

Die Erzeugung von Fließtext ist zwar die häufigste Anwendung der OM, aber nicht die einzige. So wäre es beispielsweise auch denkbar, die im K-Modell enthaltene Beschreibung einer Adaption in einer UML-Spezifikation zu maskieren. Das K-Modell würde in diesem Fall zu einem Entwurfsmuster transformiert, das die Anordnung und Interaktion von Komponenten bestimmt, die aus anderer Quelle genauer spezifiziert werden.

Wichtig ist an dieser Stelle auch noch einmal der Hinweis, dass es sich bei der OM nicht um eine 1:1 Modelltransformation handelt, sondern die K-Modellbeschreibung in ein bestehendes Modell mit zusätzlichem Informationsgehalt hineinintegriert wird und darin aufgeht.

Syntaktische Transformationen

Neben den medialen Transformationen für jeden Interaktionsschritt der Benutzerschnittstelle der Kalibrierung kann auch die inhaltliche Form der Elemente der Beschreibungstechnik aus 6.4 verändert werden.

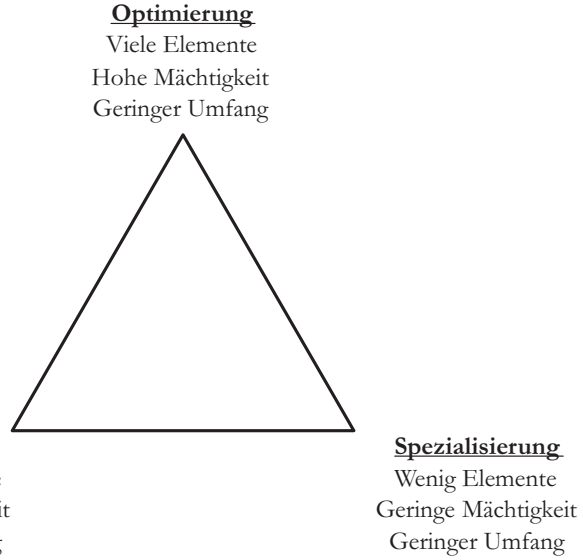


Abb. 73: Zielkonflikte bei syntaktischer Transformation der Beschreibungstechnik

Syntaktische Transformationen erlauben die Einschränkung (durch Projektion) oder Erweiterung (durch Kombination) der nutzbaren Elemente und dienen damit hauptsächlich der Anpassung eines Zielkonfliktes aus einer geringen Anzahl zu lernender Elemente, deren Mächtigkeit und des Umfangs und der Übersichtlichkeit der entstehenden Beschreibungen (Abb. 73) an die Anforderungen einer konkreten Anwendung und ihrer Benutzer.

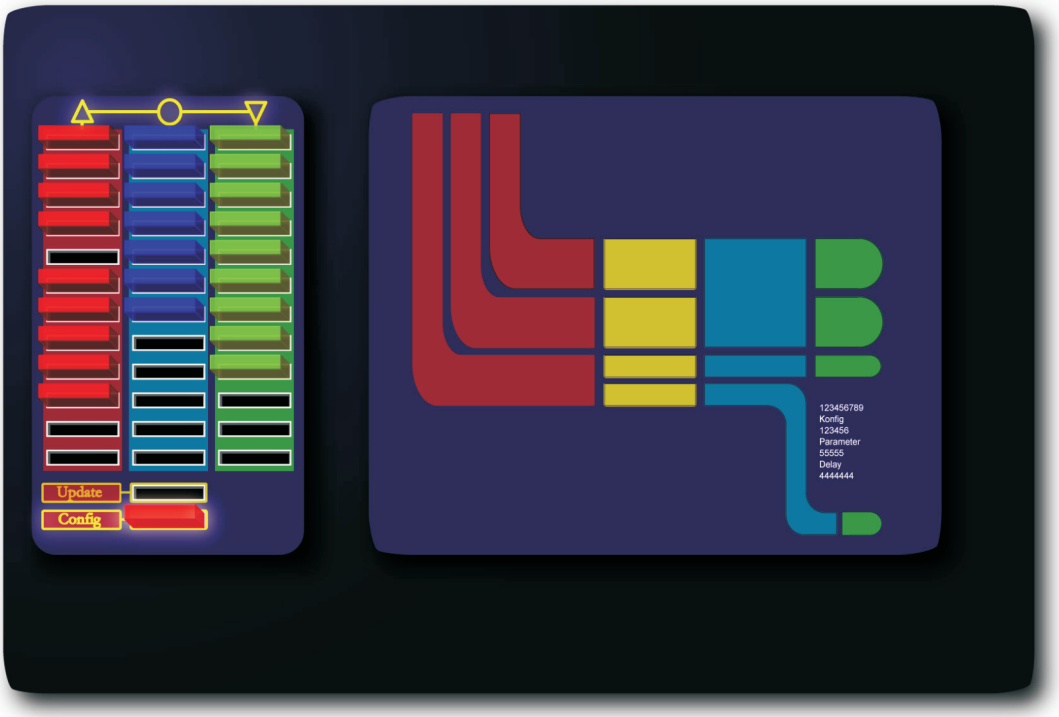


Abb. 74: HW/SW-Kalibrierungsschnittstelle der Smarthome Anwendung

Ein Beispiel für die syntaktische Anpassung ist eine Reduktion auf drei Kombinationselemente. Sensoren werden mit ihrem Ausgabekontext kombiniert, Interpreter treten nur in Form des Situationselementes auf und Aktionen werden mit ihren Aktuatoren kombiniert. Außer in kombinierter Form treten keine freien Kontextelemente auf und die Kombinationsmöglichkeiten werden darauf beschränkt, n Sensoren und m Aktionen über eine Situation zu verknüpfen.

Das Ergebnis ist eine starke Spezialisierung mit eingeschränkter Ausdrucksmächtigkeit aber guter Übersichtlichkeit, wie sie etwa für die Benutzerschnittstelle einer Kalibrierung in Form einer Stecktafel, vergleichbar einem Sicherungskasten, eingesetzt wird (Abb. 59 und Abb. 74).

Je nach Art der verwendeten syntaktischen Transformationen (siehe Abb. 75 für ein anderes Beispiel aus einem ubiquitären Gebäudeinformationssystem) und dem genauen Anwendungsgebiet können die für eine Kalibrierung notwendigen Darstellungen und Änderungen einen großen Umfang erreichen. Dieser ist dann möglicherweise nicht mit den in einem konkreten ubiquitären System temporär oder permanent zur Verfügung stehenden Wiedergabe- und Eingabemöglichkeiten zu bewältigen. In solchen Fällen kommen dann die im letzten Abschnitt beschriebenen Maßnahmen zur Ausschnittsbildung und Navigation zwischen den Ausschnitten zum tragen.

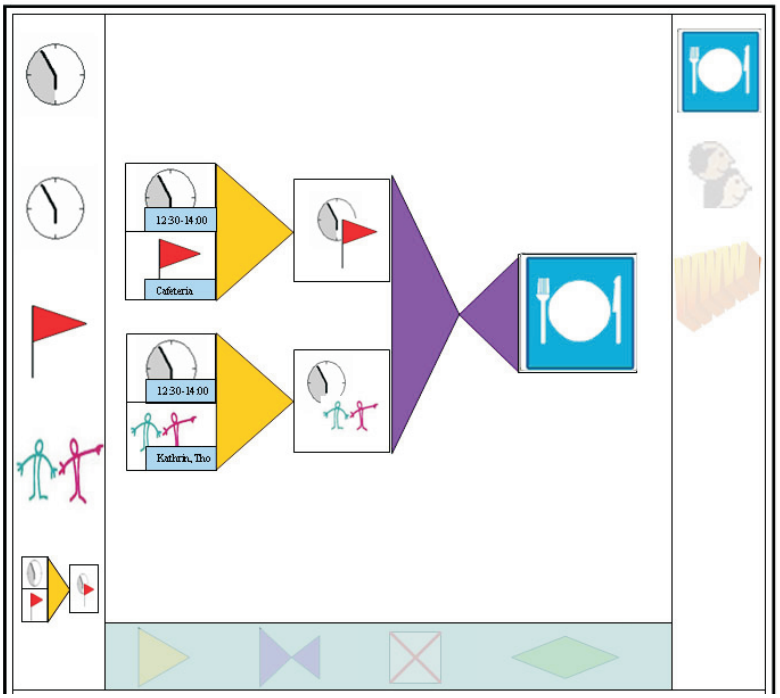


Abb. 75: Kalibrierungsschnittstelle eines Gebäudeinformationssystems

7.4 Einbettung in ein Vorgehensmodell

Wie bereits eingangs beschrieben, ist die in diesem Kapitel beschriebene Methodik kein vollständiges Vorgehensmodell für den Entwurf adaptiver Anwendungen. Es handelt sich vielmehr um einzelne methodische Bausteine, die verschiedenen Aktivitäten eines übergeordneten Vorgehensmodells zugeordnet werden können. Welches Vorgehensmodell dabei im Einzelnen für den Entwurf einer adaptiven Anwendung geeignet ist, hängt aber weniger von der Adaptivität sondern von vielen anderen Faktoren wie beispielsweise der Größe des Projektes ab. Die Auswirkungen der Adaptivität erstrecken sich dagegen meist auf technische Belange wie das Bedienkonzept, die Architektur und Realisierung der Anwendung. Eine Ausnahme davon sind die Teile des Vorgehensmodells, die später in Form der Kalibrierung auch für den Endbenutzer zugänglich gemacht werden.

Die folgenden Abschnitte geben daher lediglich eine Hilfestellung, wie die Entwurfsmethoden für die Adaptivität auf die einzelnen Aktivitäten eines Softwareentwurfsprozesses abgebildet werden können.

7.4.1 Analysephase

Die in 7.1-7.3 beschriebenen Methodik-Schritte nehmen genau genommen bereits eine vorausgegangene Entscheidung für eine ubiquitäre und damit kontextadaptive Anwendung als Voraussetzung an. Bei der Einbettung in ein vollständiges Vorgehensmodell muss diese Entscheidung also in einer geeigneten Aktivität reflektiert werden. Im Falle einer positiven Entscheidung kommt es zur Anwendung der Kontext-Engineering Methode aus 7.1. Dies bedeutet, dass zu allen ermittelten Anforderungen gleichzeitig die Umgebungseinflussgrößen ermittelt und festgehalten werden müssen, um daraus später einen Kontext konstruieren zu können. Die Entscheidung über die Ubiquität und damit die explizite Behandlung von Adaptivität sollte daher möglichst früh im Analyseprozess fallen. Andernfalls muss dieser in einer zweiten Phase erneut durchlaufen werden, um die Einflussgrößen zu analysieren.

In der Regel ist eine prinzipielle Entscheidung für oder gegen Ubiquität sehr leicht mithilfe einiger weniger Schlüsselfragen zu treffen (Abb 76).

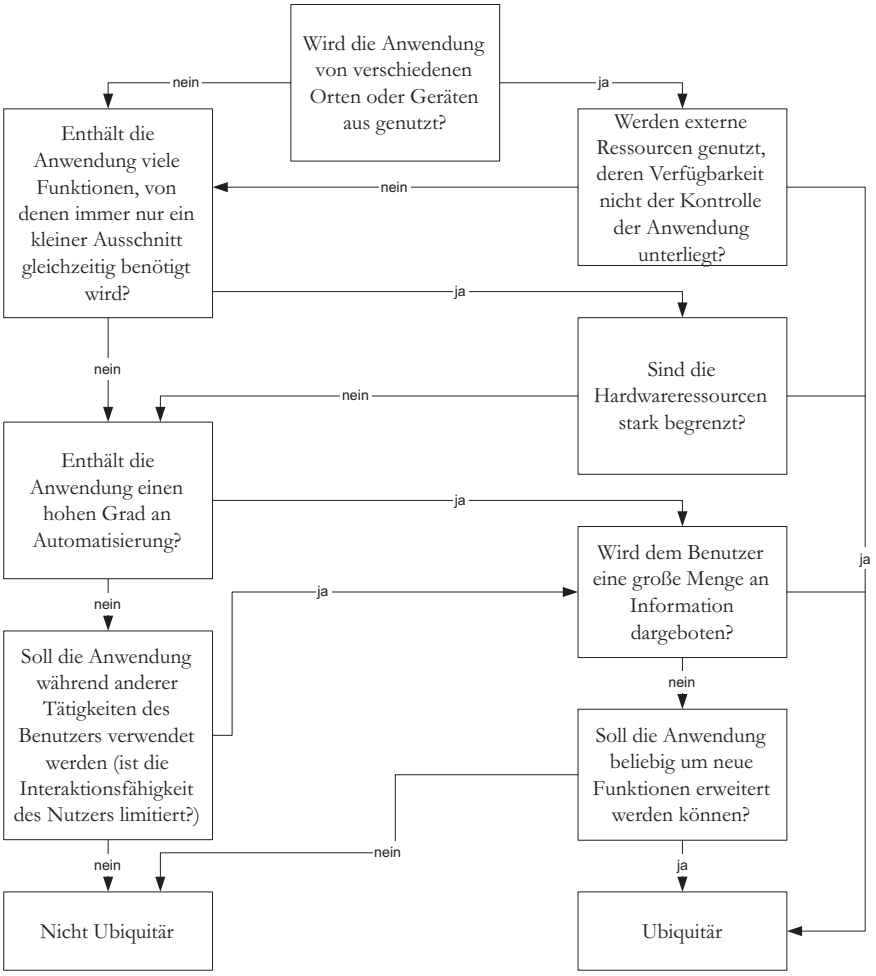


Abb. 76: Entscheidung für oder gegen Ubiquität und damit explizite Adaptivität

Besteht dagegen in Ausnahmefällen eine erhebliche Unsicherheit bei der Beurteilung der allgemeinen Rahmenbedingungen, sollte man in der Analyse zunächst von einer ubiquitären Anwendung auszugehen und eine endgültige Entscheidung erst am Ende der Analysephase treffen. Anhand der dann detailliert vorliegenden Anforderungen sollte die Entscheidung anhand des Schemas aus Abb 76 keine Probleme mehr bereiten. Als zusätzliches Merkmal kann die Adaptivitätsmetrik aus 7.1.4 herangezogen werden. Werte über 1,0 lassen sich auch als die Anzahl voneinander weitgehend unabhängiger Betriebsmodi interpretieren. Ein Betriebsmodus besteht in der Analysephase aus einer Gruppe von Anforderungen, die mit einer bestimmten Situation verknüpft sind und die sich von Anforderungen in anderen Situationen abheben oder mit diesen widersprechen können. Manuelle Wechsel durch den Benutzer zwischen mehr als zwei Betriebsmodi sind dabei eher unüblich (z.B. Anfänger/Experte, mobil/stationär etc.). Kann sich überdies die Adaptivität zur Laufzeit verändern, weil sich beispielsweise durch Personalisierungsan-

Entwurfskriterien pro oder contra Ubiquität.

forderungen neue Verknüpfungen zwischen Situationen und Anforderungen hinzufügen lassen sollen, ist dies ein sicheres Kriterium für eine ubiquitäre Anwendung.

Ist die Entscheidung für Ubiquität getroffen, müssen als Folge davon alle Aktivitäten der Anforderungsanalyse des jeweiligen Vorgehensmodells um die in 7.1 beschriebene Methodik des Kontextengineering ergänzt werden. Dies bedeutet, dass zu allen analysierten Anforderungen die jeweiligen Einflussgrößen (wann und unter welchen Umständen ist ein bestimmter Bedarf vorhanden) ermittelt werden müssen. Im Anschluss werden daraus die Abhängigkeitsbeziehungen (Situationen) zwischen den Benutzerbedürfnissen respektive technischen Anforderungen und den Umgebungsinformationen konzeptionell modelliert und validiert.

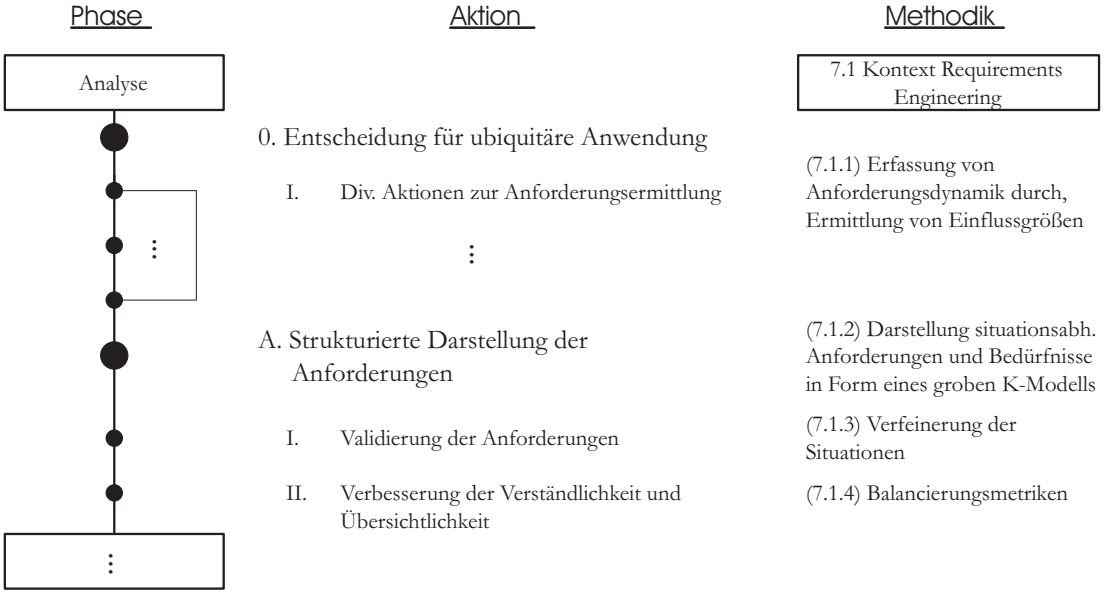


Abb. 77: Einbettung des Kontext Req. Eng. in ein gegebenes Vorgehensmodell

Abb. 77 zeigt ein Schema, wie sich prinzipiell die einzelnen methodischen Schritte für den Entwurf von Kontextadaption auf die Aktionen eines fiktiven Vorgehensmodells abbilden lassen.

7.4.2 Designphase

Den größten Einfluss auf ein Vorgehensmodell hat die in 7.1-7.3 beschriebene Entwurfsmethodik für Kontextadaption in der Designphase. Da die Methodik auf der Realisierung der konzeptionellen Idee der Kontextadaption und ihrer Kalibrierung in Form eines Framework basiert, sind durch dieses bereits große Teile der Architektur einer ubiquitären Anwendung festgelegt. Dabei muss man zwischen der Architektur des Gesamtsystems und der Ar-

chitektur einzelner darin enthaltener Komponenten unterscheiden. Durch das Framework ist im Wesentlichen ersteres betroffen, da die Architektur des Gesamtsystems durch die im Framework vorgenommene Unterteilung in die zwei Subsysteme Kontextadaption und Anwendungskern sowie deren Zusammensetzung aus Sensor-, Interpreter- und Aktuatordiensten dominiert wird. Freiheitsgrade bestehen lediglich in der Entscheidung, ob ein Dienst an eine externe oder interne Komponente gebunden wird.

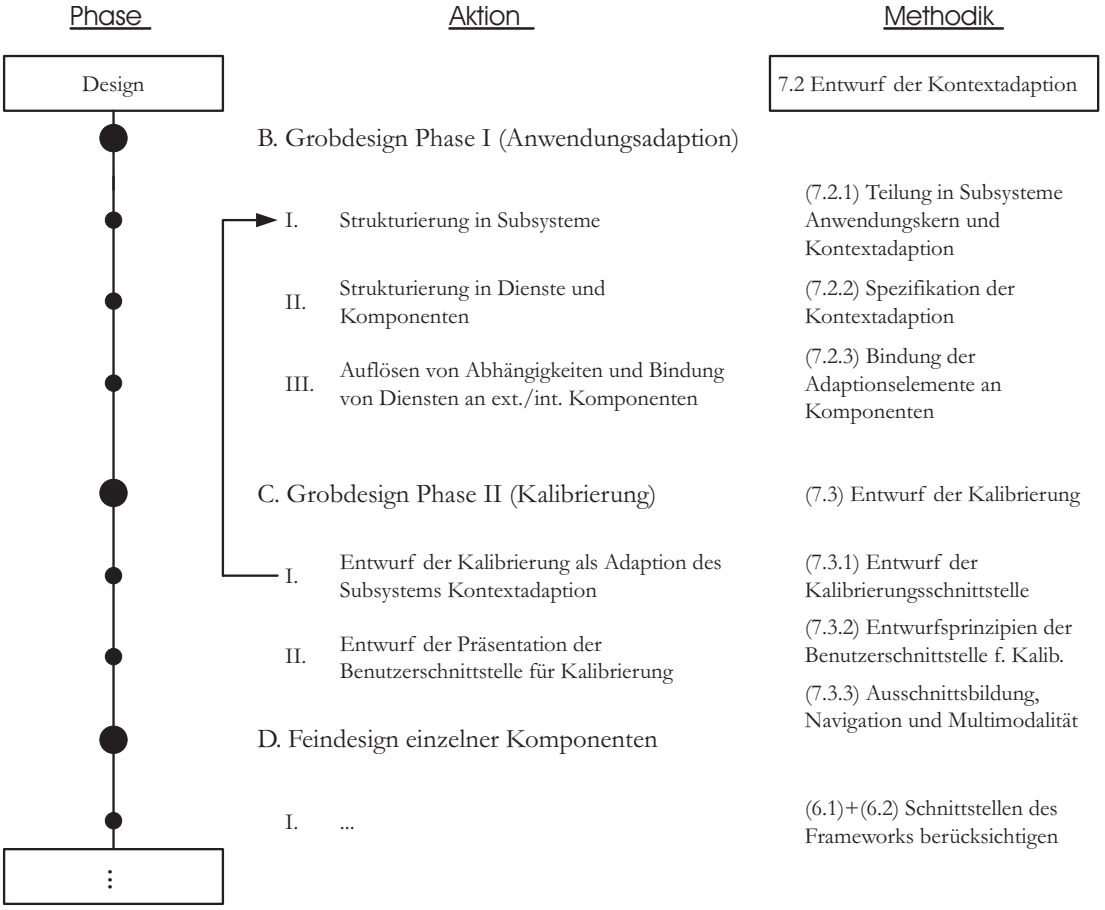


Abb. 78: Einbettung der Entwurfsmethodik in ein gegebenes Vorgehensmodell

Die Festlegung, ob für die Komponente weitere Dienst- oder Komponentenabhängigkeiten bestehen, also die Architektur einzelner Komponenten ist davon aber wiederum unbeeinflusst. Bei externen Komponenten liegt die Realisierung zudem außerhalb der Kontrolle des Anwendungssystems. Das bedeutet, es wird von einer (umgebungsabhängigen) Existenz einer Komponente ausgegangen, die außerhalb des Vorgehensmodells entwickelt wurde.

Die Aktivitäten des Vorgehensmodells zum Feindesign einzelner Komponenten verlaufen also weitgehend unbeeinflusst. Durch das Framework beziehungsweise deren Adaptoren sind lediglich für einige Komponenten be-

stimmte Schnittstellen vorgegeben. Eine Ausnahme ist die Komponente (oder Komponenten) zur Realisierung des Kontextes. Hier muss ein Anfangszustand spezifiziert werden, der einen Teil der gerade erstellten Designspezifikation in Form eines K-Modells beschreibt.

Bezüglich des Vorgehensmodells bietet es sich daher an, alle konstruktiven Aktivitäten des Grobdesigns bereits von Anfang an auf die Konstruktionsmethodik aus 7.2 abzubilden. Das Feindesign einzelner Komponenten bleibt davon methodisch unberührt. Die Konstruktionsschritte der Adaption, Kalibrierungsschnittstelle und Benutzerschnittstelle der Kalibrierung werden einfach auf mehrere Phasen oder Iterationen des Grobdesigns abgebildet (Abb. 78).

7.4.3 Realisierungsphase

Die Einbettung der in 7.1-7.3 beschriebenen Methodik in die Realisierungsphase eines gegebenes Vorgehensmodell beschränkt sich im Wesentlichen auf die Art der Realisierung einzelner systeminterner Komponenten:

- Das Framework aus Kapitel 6 stellt im Zusammenhang mit der dort definierten Referenzarchitektur die Realisierung einiger Infrastrukturkomponenten oder generisch wiederverwendbarer Implementierungen bereit.
- Die generische Kontextkomponente muss mit der im Design entwickelten K-Modellspezifikation initialisiert werden.
- Die durch das Framework standardisierten Zugriffsschnittstellen auf Kontextinformationen stehen für die anwendungsspezifischen Sensor-, Interpreter- und Aktuatorkomponenten in Form von ableitbaren Basisklassen bereit.

Aufgrund der Flexibilität und Dynamik des sich ergebenden Gesamtsystems sind einige der üblichen Test- und Qualitätssicherungsverfahren für ubiquitäre Anwendungen nicht ausreichend. Entsprechende Aktionen im gegebenen Vorgehensmodell müssen daher auf neue Methoden abgebildet werden. Eine genauere Betrachtung von Testmethoden für adaptive Systeme würde allerdings den Rahmen dieser Arbeit sprengen. Daher bleibt es an dieser Stelle bei dem Hinweis, dass sich zum Testen am besten speziell modifizierte Versionen der Kalibrierung und des in 7.3.2 beschriebenen SUB-Monitorings eignen. Diese richten sich dann allerdings nicht mehr an den Endbenutzer zur Laufzeit und können daher an einigen Stellen ohne Beachtung der Ubiquität oder

Systemintegrität sowie mit detaillierteren technischen Informationen operieren.

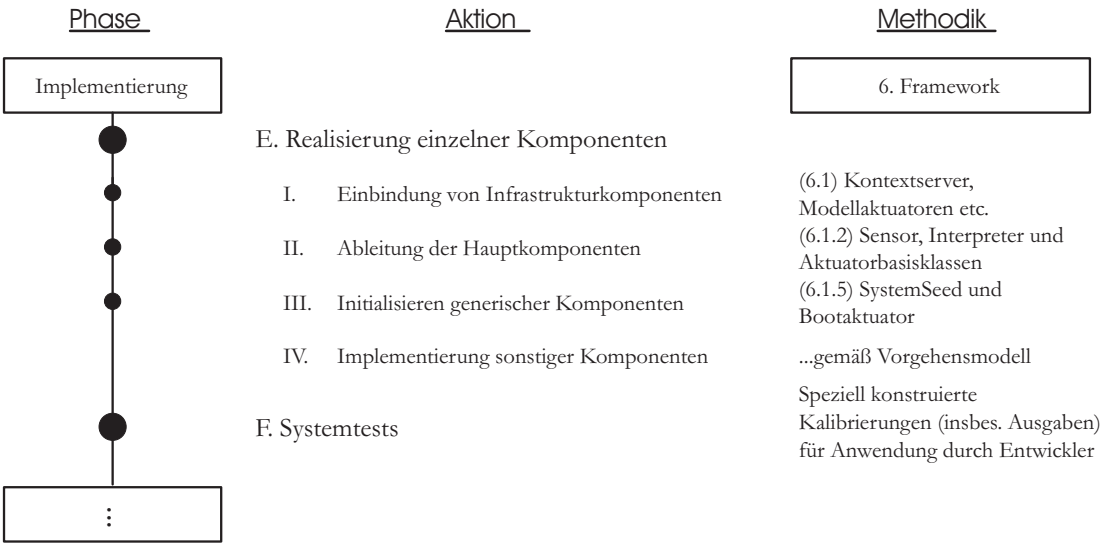


Abb. 79: Einbettung des Framework in ein gegebenes Vorgehensmodell

KAPITEL 8 Schlussbewertung der praktischen Bedeutung

Wie lässt die praktische Bedeutung der kalibrierbaren Kontextadaption bewerten?

Das letzte Kapitel dieser Arbeit zieht ein Resümee über die verschiedenen Eigenschaften und experimentell gewonnenen Erfahrungen mit der Verwendung von Kalibrierung und des in Kapitel 6 beschriebenen Framework.

8.1 Erfüllung der Anforderungen in Prototyp II

Im Rahmen der in Kapitel 2 beschriebenen Fallstudie wurde noch ein weiterer Prototyp der dort beschrieben ubiquitären Smarthome-Anwendung realisiert. Ziel dabei war die Beurteilung der Praxistauglichkeit der in dieser Arbeit vorgestellten kalibrierbaren Kontextadaption und deren Umsetzung in einem technischen und methodischen Framework sowie die Validierung der daran in Kapitel 4 gestellten Anforderungen.

Zu diesem Zweck wurden die für den ersten Prototyp implementierten Grundfunktionen einfach als Kernsystem in den in Kapitel 7 beschriebenen Entwurfsprozess übernommen. Im Anschluss daran wurde wie aus den entsprechenden Beispielen in Kapitel 7 ersichtlich das K-Model der Kontextadaption konstruiert, mit einer Kalibrierungsschnittstelle ausgestattet und eine Benutzerschnittstelle für die Kalibrierung entworfen. Zu Testzwecken wurden Teile dieser Benutzerschnittstelle zudem implementiert.

8.1.1 Kalibrierbarkeit

Aus den beiden Ubiquitätsbedingungen (Nutzbarkeit und Verfügbarkeit in möglichst vielen Situationen) ergaben sich drei wichtige Anforderungen an die zudem möglichst vollständige Reprogrammierbarkeit der Kontextadaption in Form der Kalibrierung. Zum einen muss die Kalibrierung zur Laufzeit erfolgen können, zum anderen eine vollständige Reprogrammierung der Kontextadaption erlauben. Aufgrund der zweiten Ubiquitätsbedingung muss die Kalibrierung zudem eine Möglichkeit bieten, verschiedene technische Realisierungsverfahren der Kontextadaption in einheitlicher Weise zu manipulieren. Dabei muss auch gewährleistet sein, dass die Kalibrierung eine Migration des Kalibrierungsverfahrens von einem auf ein anderes technisches Verfahren gewährleistet. Es ist also nicht damit getan, einen einzigen Regelmechanismus zu entwerfen. Dieser könnte mit bestimmten Aufgaben überfordert oder in bestimmten Hardwaresituationen nicht einsetzbar sein.

Vollständige Reprogrammierbarkeit der Adaption.

Alle drei Anforderungen werden durch die formale Definition der kalibrierbaren Kontextadaption in Kapitel 5 und deren technische Umsetzung als Framework (Kapitel 6) erfüllt. Eine vollständige Reprogrammierung ist durch die in 5.3 beschriebene Möglichkeit zur Selbstkalibrierung, also der Formulierung der Kalibrierung als weiterer kalibrierbarer Kontextadaption gegeben (siehe auch Filterkomponente F_{A4} in 5.3). Dass die Änderungsmöglichkeiten vollständig sind, macht man sich auch leicht durch die folgende technische Überlegung klar. Das die Adaption einer Anwendung bestimmende K-Modell wird als Spezifikation in deren Kontext abgelegt und über einen speziellen Modellaktuator ausgewertet. Dieser bildet die Spezifikation auf eine dynamische Implementierung auf der Basis verfügbarer Komponenten ab. Dies geschieht unter Nutzung von unterhalb des Framework angesiedelter Middleware und Betriebssystemfunktionen. Der Modellaktuator ist zudem in der Lage, sich selbst durch eine andere Implementierung zu ersetzen. Invarianten einer solchen Änderung sind also lediglich die Schnittstellenspezifikationen des Framework und natürlich die Realisierungen der darunter liegenden Betriebsystemfunktionen. Über einen Zwischenschritt kann aber theoretisch auch das Adaptionsmodell einer Anwendung durch den Modellaktuator auf eine andere Plattform übertragen und dort in den Kontext einer Anwendung überspielt werden, die mit einem anderen Framework und einem anderen Betriebssystem realisiert wurde.

Die Erfüllung der zweiten Anforderung nach einer Durchführbarkeit der Kalibrierung zur Laufzeit ergibt sich ebenfalls aus der formalen Definition der Kalibrierung in 5.3. Der Filter F_{A4}, der die Kalibrierung dort modelliert, wird über einen Eingabekanal (also zur Laufzeit) kontrolliert. Das Argument lässt sich auch wiederum technisch veranschaulichen. Die Beschreibung der Adap-

tionslogik ist im Kontext der Anwendung abgelegt und kann dort wie jeder andere Wert verändert werden. Der Modellaktuator (als Implementierung der Filterfunktion F_{A4}) wird bei einer Änderung benachrichtigt und rekonfiguriert das die Kontextadaption implementierende System einschließlich von sich selbst gemäß dieser Vorgaben. Dass bei der Rekonfiguration zur Laufzeit keine inkonsistenten Systemzustände entstehen, wird durch die Entkoppelungseigenschaft sichergestellt. Die Elemente eines K-Modells kommunizieren niemals direkt, sondern sind immer über den Kontext entkoppelt. Diese Eigenschaft wurde im Entwurf des Frameworks auch auf die Komponenten übertragen, welche die Kontextadaption realisieren. Eine Rekonfiguration zur Laufzeit ist damit solange konsistent, wie der Kontext von einer Rekonfiguration ausgeschlossen bleibt. Soll der Kontext ebenfalls rekonfiguriert werden, beispielsweise um das Gesamtsystem auf eine andere Plattform zu übertragen, müssen zusätzliche Sicherungsmaßnahmen getroffen werden. Diese können aber in Form einer zusätzlichen Adaptionslogik in das System eingebracht werden, zum Beispiel für die Erkennung zusammengenutzter Cluster von Kontextelementen, das Sperren und Freigeben von Kontextelementen, die Sicherung von Transaktionen etc.

Kalibrierung zur Laufzeit.

Die dritte Anforderung nach einer Einbindung und Migration unterschiedlicher Kalibrierungsverfahren wie etwa Konfigurationsdialoge, Editoren oder lernfähige Lösungen ist durch die Art der Konstruktion der K-Modelle gegeben. Die einzelnen Dienste, die ein K-Modell realisieren, können zur Laufzeit auf andere Implementierungen abgebildet werden. Da die Kalibrierung nichts anderes ist, als eine spezialisierte Adaption für die Komponenten, welche die Kontextadaption realisieren, gilt diese Rekonfigurationsmöglichkeit natürlich auch für die Kalibrierung. So kann beispielsweise die Implementierung eines Interpreters, der eine Kalibrierung durchführt, ohne weiteres zur Laufzeit von einem Editor hin zu einer Sprachdialogsteuerung oder einem neuronalen Netz gewechselt werden, solange diese Implementierungen auf den gleichen Kontextdaten arbeiten können. Selbst wenn dies nicht der Fall ist, kann immer noch eine Kalibrierung durchgeführt werden, welche die Gesamtstruktur der Adaption so verändert, dass über entsprechende zusätzliche Interpreter-Kontextelement-Ketten eine Transformation der Kontextinformationen in das für die neue Kalibrierungsimplementierung benötigte Format durchgeführt wird.

Implementierungsunabhängige Kalibrierung.

8.1.2 Nebenläufigkeit

Die Anforderung nach einer Möglichkeit zur Synchronisation von nebenläufigen Kontextadaptionen allgemein und Kalibrierungen im Speziellen wer-

den durch das in dieser Arbeit vorgestellte Adaptionskonzept in zwei Schritten erfüllt.

Die Eingabeexklusivitätsregel für Kontextelemente gewährleistet, dass Kontextinformationen immer nur durch eine einzige Quelle verändert werden dürfen. Dies ist notwendig, da alle Elemente des in dieser Arbeit verwendeten K-Modells so spezifiziert sind, dass sie prinzipiell nebenläufig sind. Sollen unterschiedliche Informationen zusammengeführt werden, dann geschieht das ausschließlich über einen dafür konstruierten Interpreter. Dient ein solcher Interpreter der Zusammenführung der Ergebnisse zweier ansonsten unabhängiger Adaptionsstränge, ist durch ihn ein fallspezifischer Synchronisationsmechanismus definiert.

Solche fallspezifischen Synchronisationsinterpreter können auch dazu benutzt werden, das in 4.2.1 genannte Propagationsproblem zu lösen. Dies geschieht, indem parallel zu den nebenläufig adaptierten aber von einander abhängigen Schichten[1] eine weitere Adaption eingeführt wird, welche die beiden Änderungen über ein gemeinsames Kontextelement synchronisiert (siehe Abb. 80).

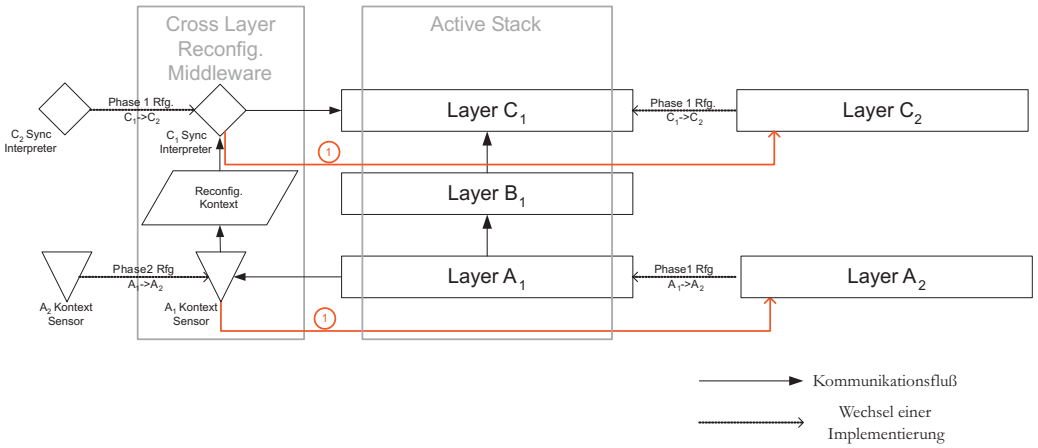


Abb. 80: Prinzip der Synchronisation nebenläufige Kalibrierungen

Je nach Aufwand kann damit das Propagationsproblem verringert oder auch ganz beseitigt werden. Das Gleiche gilt auch für das Problem der Propagationsunterbrechung bei einer Rekonfiguration. Durch die im Framework und in der Methodik vollzogene Trennung der Anwendung in die zwei Subsysteme des Anwendungskerns und seiner Kontextadaption ist es möglich, die für

[1] Das Problem wurde aus Gründen der Einfachheit für lineare Abhängigkeiten, also Schichten erklärt. Problem und Lösung sind aber auch für nichtlineare Abhängigkeiten gültig.

die Synchronisation notwendige Kommunikation während der Rekonfiguration des Kernsystems im Adaptionssubsystem aufrecht zu erhalten. In einer zweiten Phase (Abb. 80) kann dann das Adaptionssystem selbst an die Synchronisationsbedürfnisse des adaptierten Anwendungskerns angepasst werden. Da das Adaptionssubsystem die Rekonfigurationen des Kernsystems kontrolliert, sind während dessen weitere Rekonfiguration des Kerns unterbrochen. Aufgrund der Entkoppelungseigenschaft ist diese Unterbrechung aber nur auf den indirekt von den geänderten Kontextadaptionselementen betroffenen Teil des Kernsystems beschränkt.

8.1.3 Intuitive Beschreibung und Benutzerschnittstelle

Von allen Anforderungen ist eine objektive Beurteilung der Intuitivität und Benutzerfreundlichkeit die schwierigste. Ohne zu Hilfenahme einer größeren Anzahl menschlicher Versuchspersonen ist in diesem Punkt kaum eine repräsentative Bewertung zu erreichen. Zudem hängt die endgültige Beurteilung auch von der jeweiligen Umsetzung in einer konkreten Anwendung ab.

Framework und Methodik stellen aber eine Reihe von Techniken zu Verfügung, das eigentliche Grundmodell der Kontextadaption und seiner Kalibrierung in fast jede beliebige mediale, syntaktische oder Symbolisierungsform zu transformieren, einschließlich der Möglichkeit für multimodale Darstellungsformen, Schnittstellen und beliebige Grade an Automatisierungskonzepten. Diese Flexibilität ermöglicht zumindest die potenzielle Erfüllung der genannten Anforderungen.

Die für den Prototyp der Fallstudie genannten Kalibrierungskonzepte wurden dagegen nur an einigen kooperativen Versuchspersonen erprobt. Die Ergebnisse waren dabei wie erwartet uneinheitlich. Abhängig von persönlichen Vorlieben und Grad des technischen Verständnisses wurden je nach Person unterschiedliche Kalibrierungsformen bevorzugt. Diese Grundeinstellung wurde aber auch von der jeweiligen Testsituation beeinflusst.

Dieses Ergebnis passt zu der Vermutung, dass eine "beste" Art der Kalibrierung nicht existiert, sondern wie die Adaption der eigentlichen Anwendung an die jeweilige Situation und der damit verbundenen persönlichen Vorlieben des Benutzers kontextabhängig angepasst werden muss.

8.1.4 Effizienz

Obwohl nicht explizit in den Anforderungen von Kapitel 4 genannt, spielt natürlich auch die Effizienz des in dieser Arbeit beschriebenen Framework und der zugrunde liegenden Konzepte eine Rolle. Diese wurde im Rahmen eines anderen Projektes mithilfe der in [94] beschriebenen Evaluierungsme-

thoden genauer untersucht. Erste Ergebnisse dazu wurden in [95] veröffentlicht. Abb. 81 zeigt als Beispiel daraus die Adaptionszeit in ms gegenüber der Anzahl gleichzeitiger Adaptionen in einem Mehrbenutzer Szenario aus einer Telekommunikationsanwendung.

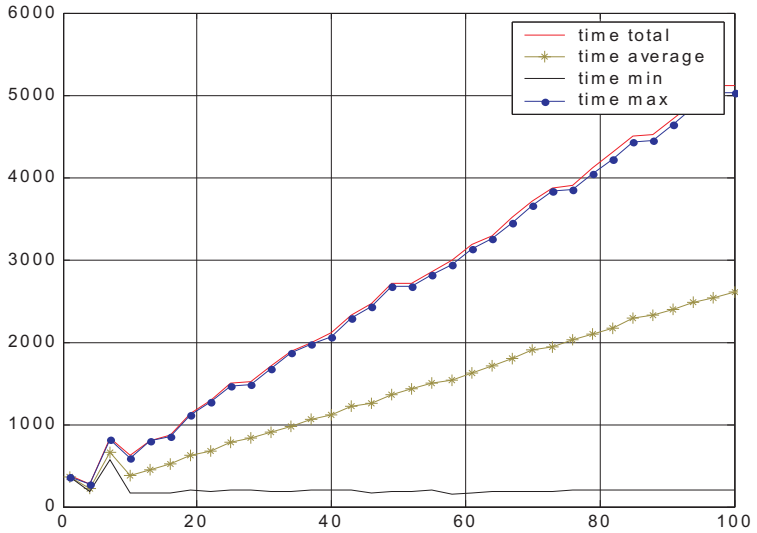


Abb. 81: Adaptionszeit vs. simultane Adaptionen in einem verteilten 10Mbit Netz

Obwohl die Implementierung des Framework noch prototypischen Charakter hat und noch keine der beispielsweise in [103] beschriebenen Optimierungskonzepte realisiert wurde, zeigen die Messungen bereits eine Leistung, die etwa vier mal besser als die vergleichbarer Prototypen[68] ist und das trotz der Möglichkeit der Kalibrierung der Adaption, die in anderen Framework nicht gegeben ist.

8.1.5 Effektivität der Kalibrierung

Die Wirksamkeit der Kalibrierung von Kontextadaption mithilfe des Framework und der Methodik dieser Arbeit wurde an den für die Fallstudie ermittelten SUB Ereignissen untersucht. Dabei wurden zwei Fälle unterschieden:

- Weiteres Auftreten eines registrierten SUB kann durch eine Kalibrierungsmaßnahme verhindert werden.
- Das jeweilige SUB kann über die für den Prototyp entworfene Kalibrierungsschnittstelle im Voraus durch den Benutzer erkannt werden.

Dabei besitzt der zweite Fall natürlich nur beschränkte Aussagekraft darüber, ob durch bestimmte allgemeine Kalibrierungsschnittstellen das SUB von vornherein hätte erkannt werden können. Ganz im Gegenteil lässt die Natur

des Frame-Problems als Verursacher vermuten, dass dies im Allgemeinen nur mit einer gewissen Wahrscheinlichkeit gelingen kann. Nicht vernachlässigen darf man jedoch die Tatsache, dass es dem Benutzer über das Mittel der Kalibrierung auch möglich ist, neue (ausgabeorientierte) Kalibrierungsmöglichkeiten hinzuzufügen. Der Sinn einer solchen Kalibrierung zur zukünftigen Erkennung eines bereits einmal aufgetretenen SUB liegt natürlich darin, ein erneutes Auftreten im Voraus zu erkennen, um rechtzeitig Maßnahmen zu seiner Verhinderung zu ergreifen. Sinnvoll ist ein solches Vorgehen immer dann, wenn sich ein registriertes SUB nur schwer durch eine einmalige Kalibrierung verhindern lässt, oder wenn sich durch die Erkennung zuverlässig eine ganze Gruppe von SUB-Phänomenen abdecken lässt, die Beseitigung aber eine für den Einzelfall spezialisierte Kalibrierung erfordert.

Daraus kann auch eine einfache Kalibrierungsstrategie abgeleitet werden. Wurde ein SUB registriert, sollte der Benutzer zunächst eine Kalibrierung planen, welche die Früherkennung zukünftiger ähnlich gelagerter SUBs ermöglicht. Erst im zweiten Schritt wird dann eine Kalibrierung durchgeführt, die das konkrete SUB in Zukunft verhindert.

Bei der Bewertung der Effektivität des Kalibrierungskonzeptes allgemein wurde auf ähnliche Weise vorgegangen. Für jedes der in 2.4.3 genannten SUB des Prototyps wurde untersucht, ob sich erstens eine Kalibrierung finden ließ, welche eine Vorauserkennung ähnlich gelagerter Fälle ermöglicht und zweitens ob es gelingt, das konkrete SUB durch eine Kalibrierung zu beseitigen. Bei den Wirksamkeitsuntersuchungen stellte sich aber auch heraus, dass die Kalibrierung für eine Früherkennung dem Benutzer in der Regel mehr technisches Detailverständnis und auch die Verwendung komplexerer Benutzerschnittstellen für die Kalibrierung abverlangt. Dem kann man aber durch eine größere Auswahl an vorgefertigten Kalibrierungsmaßnahmen begegnen, die der Benutzer dann lediglich im Rahmen seiner Kalibrierung hinzufügen oder entfernen kann. Das muss auch nicht unbedingt ein technischer Prozess sein. Der Entschluss, vor ungewöhnlichen Situationen öfter mal einen Blick in das Handbuch zu werfen kann eine ebenso effektive Kalibrierungsmaßnahme für SUB Früherkennung sein, wie das Hinzufügen eines neuen Softwaremoduls.

Intransparente Ressourcen

Gerade bei der Kalibrierung von SUB, die durch eine intransparente Ressourcenlage entstehen, spielt die (Voraus-)Erkennung oft eine größere Rolle bei der Kalibrierung als deren Verhinderung, weil diese trivial ist, sobald die wesentlich schwierigere Erkennbarkeit gegeben ist.

Die beiden in 2.4.3 genannten SUB Beispiele dieser Kategorie bezogen sich nämlich immer auf eine Situation, in welcher der durch den Benutzer erkennbare Ressourcenkontext und seiner Eigenschaften von dem für das System tatsächlich verfügbare Ensemble aus Hard- und Software abwich.

Die SUBs dieser Kategorie entstehen also im Unterschied zu den meisten anderen Kategorien nicht dadurch, dass das Modell des Systems nicht mit der Wirklichkeit übereinstimmt, sondern weil die Wahrnehmung des Benutzers nicht mit der Wirklichkeit übereinstimmt und er deshalb die falschen Erwartungen an das Systemverhalten stellt. Dieser Umstand wäre leicht zu beheben, wenn das System in der Lage wäre, zu erkennen, dass die Wahrnehmung des Benutzers nicht mit der Wirklichkeit übereinstimmt. Für diese erweiterte Aufgabe sind die dem System vorliegenden Informationen über die Wirklichkeit aber zu ungenau.

Im Beispiel der nicht erfolgreichen drahtlose Verbindung mit dem Mobiltelefon konnte das System zwar richtig erkennen, dass das Gerät nicht zur Verfügung steht und adaptiert dahingehend richtig, indem die entsprechende Funktion nicht ausgeführt wurde. Dem würde auch der Benutzer zustimmen, wenn er wüsste, dass keine Verbindung zustande kam. Der Benutzer ging aufgrund der physischen Anwesenheit des Gerätes aber zunächst von einer erfolgreichen Verbindung aus und erwartete daher fälschlicherweise eine andere Adaption und registrierte daher ein SUB. Leider war die falsche Annahme des Nutzers von der Anwendung nicht zu erkennen, da für sie keine Möglichkeit bestand, zwischen der Abwesenheit und der Funktionsuntüchtigkeit eines drahtlos verbundenen mobilen Gerätes zu unterscheiden.

Für die Kalibrierung ergeben sich daraus zwei mögliche Ansätze. Zum einen ist das die Durchführung einer Kalibrierung welche es dem System erlaubt, richtig zwischen Abwesenheit und Funktionsuntüchtigkeit einer Ressource zu unterscheiden und den Benutzer in einem solchen Fall zu informieren. Dies wird allerdings lediglich in spezifischen Einzelfällen funktionieren.

Für das in 2.4.3 beschriebe SUB Problem mit den batteriebetriebenen Bewegungsmeldern kann eine solche Kalibrierung durch den Benutzer durchgeführt werden. Dazu ist es notwendig, über die in 7.3.2 beschriebene Kalibrierungskonsole und den dort beschriebenen Editor, eine neue Adaption hinzuzufügen.

Diese besteht aus einem vom Framework bereits zur Verfügung gestellten Interpreter, welcher aus dem Metamodell die letzte Änderung eines Sensorwertes extrahiert und mit einem vorgegebenen Wert vergleicht. Der Ausgabekontext dieses Interpreters muss nur noch mit einem entsprechenden Notifikationsaktuator verbunden werden. Eine solche Kalibrierung funktioniert natürlich nur im Einzelfall, etwa wenn angenommen werden kann, dass der zum Bewegungsmelder gehörende Raum auch tatsächlich mit einer gewissen Regelmäßigkeit betreten wird.

Eine andere Möglichkeit ist natürlich die richtige Nutzung einer bereits vorhandenen Kalibrierungsmöglichkeit.

Im Entwurf des Prototyps war für solche Zwecke bereits eine spezialisierte Benutzerschnittstelle der Kalibrierung (systeminitiierte Kalibrierungsausgabe) vorgesehen worden. Diese funktioniert allerdings nur für portable Geräte, also Geräte, die zwar im Laufe der Anwendung hinzugefügt oder entfernt werden, aber längere Zeit an ihrem Platz verweilen. Beispiele dafür sind die meisten Haushaltsgeräte. Deren Defekterkennung war in 7.3.2 Abb. 56 beschrieben worden und basiert auf der Verwendung zusätzlicher Steckmodule (ähnlich elektrischen Sicherungen) in einer speziellen Hardwarekonsole. Kann ein Gerät nicht erkannt werden, dass sich eigentlich erreichen lassen müsste, wird diese dem Benutzer auf optische oder andere Weise angezeigt.

Der zweite Ansatz für die Kalibrierung von SUB aus der Kategorie intransparenter Ressourcensituationen besteht darin, durch geeignete Kalibrierungsausgaben den Benutzer dazu in die Lage zu versetzen, zu erkennen, ob ein mobiles Gerät funktionstüchtig ist, oder nicht.

Für den Prototyp bestand eine geeignete Kalibrierung darin, das Ereignis der Netzverbindung des mobilen Gerätes aus dem Kontext abzugreifen und in ein unaufdringliches akustisches Signal zu verwandeln. Dies versetzt den Benutzer dazu in die Lage, das für ihn ansonsten nicht direkt registrierbare Ereignis der Funktionstüchtigkeit zu erkennen. Das Ausbleiben des Signals bringt den Benutzer auf den gleichen Informationsstand wie das System und verhindert dadurch die Erwartung einer nicht durchführbaren Adaption.

Erste Standardkalibrierung: Wechsel der Realisierung.

Auf eine der oben genannten Weisen konnten auch die meisten restlichen in der Fallstudie registrierten SUBs dieser Kategorie kalibriert werden. Die verbleibenden zwei schwierigen Fälle von SUB wurden dagegen über eine der

drei Standardkalibrierungen beseitigt. Die erste Standardkalibrierung besteht dabei aus dem simplen Austausch einer der Komponenten der Adaption.

> So wurde im Prototyp der unzuverlässige Sensor der Wettervorhersage einfach an einen anderen Anbieter gebunden.

Zweite Standardkalibrierung: Einschränkung der Qualität.

Funktioniert dies nicht, weil etwa keine Alternative zur Verfügung steht, kann auch die genutzte Qualität in der Adaption eingeschränkt werden.

> Die manchmal unzuverlässige Angabe von Zugverspätungen wurde dadurch kalibriert, dass die angezeigte Verspätung durch einen eingefügten Interpreter halbiert und damit die maximal mögliche Genauigkeit nicht voll ausgeschöpft wurde.

Dritte Standardkalibrierung: Einschränkung der Funktion.

Die dritte Standardkalibrierung besteht aus der zeitweisen oder permanenten Deaktivierung einer Adaption. Dies war im untersuchten Prototyp in dieser SUB-Kategorie aber nicht notwendig.

Unvorhergesehene Nutzung/Kombination

Das erste der beiden in der Fallstudie in dieser Kategorie registrierten SUBs betraf die für den Benutzer unvorhersehbaren Folgen einer Adaption. Kalibriert werden konnte dieses Phänomen durch die Möglichkeit, mithilfe des Framework für den Prototyp bestimmte Adaptionen zu simulieren respektive zu erzwingen.

> Notwendig war dies, weil die genaue Ursache des SUBs zunächst nicht ohne weiteres erkannt werden konnte. Mithilfe der in 7.3.2 beschrieben Möglichkeiten der SUB-Simulation konnte die Urlaubsadaption erzwungen und beobachtet werden. Dabei wurde festgestellt, dass die Abschaltung elektrischer Verbraucher unabsichtlich auch die Telefonanlage deaktivierte. Ein einfaches Umstecken auf eine nicht abschaltbare Steckdose brachte die Lösung.

Dieses Beispiel zeigt auch wieder, dass eine Kalibrierung nicht unbedingt aus einer Veränderung der Adaption des ubiquitären Anwendungssystems bestehen muss, sondern auch durch eine Anpassung der Umgebung oder Nutzerverhaltens möglich ist, sofern die Umstände der direkten Kontrolle des Nutzers unterstehen.

Das zweite in dieser Kategorie beschriebene SUB wurde dagegen durch das

Hinzufügen eines weiteren Sensors behoben, der die bisher unvorhergesehene Situation für das System erkennbar machte.

Die Kalibrierung für den Fall des vergessenen aber nicht ortbaren Mobiltelefons erscheint zunächst schwierig. Verschiedene Kalibrierungsversuche auf der Basis einer Parallelverfolgung der schwer trennbaren Adaptionszweige (zunächst Versuch einer Weiterleitung aber dann lokale Aufzeichnung für den Fall der Nichtannahme) scheiterten an technischen Schwierigkeiten. Die Lösung kam in Form einer neuen teilweisen Ortbarkeit des Mobiltelefons, die zum Zeitpunkt der Anwendungsentwicklung nicht zur Verfügung stand. Mit dieser neuen Lösung konnte das Betreten und Verlassen bestimmter Funkzellen das Mobiltelefon zum Senden einer SMS veranlassen. Die nachträgliche Einbindung dieses Sensors in Form einer Kalibrierung konnte dann sicherstellen, dass der Aufenthaltsort des Telefons sicherer bestimmt werden konnte. Solange das Telefon erreichbar war, aber keine SMS über das Verlassen der Heimzelle vorlag, konnte das System davon ausgehen, dass das Telefon noch vor Ort war und die Anrufe nicht weiterleiten. Sollte später ein kostengünstigerer Weg der Ortsbestimmung vorliegen, kann dieser durch eine weitere Kalibrierung eingebunden werden.

Messungenauigkeiten, Fehlinterpretationen

In günstigen Fällen können die SUB dieser Kategorie durch das Hinzufügen neuer Sensoren oder Interpretationen auf der Basis nicht allgemein gültiger Annahmen kalibriert werden. Diese werden dann entweder als Ergänzung der bestehenden Adaption in deren Sensorkontext hinzugefügt, oder beschützen als Sicherungseigenschaft deren Aktuatorkontext.

Im Fall der in 2.4.3 genannten Fehlinterpretation der Außentemperatur aufgrund besonderer thermischer Ausnahmesituationen war die Erkennung des Eingangskontextes ja bereits durch einen redundanten Temperatursensor verbessert worden. Dies war aber nicht genug, um das großflächig wirkende thermische Phänomen und das daraus folgende SUB zu verhindern. Daher wurde der Adaption durch Kalibrierung eine zusätzliche Sicherungseigenschaft hinzugefügt. Diese bestand in der Auswertung eines günstig in der Nähe des zu öffnenden Fensters platzierten Innentemperatursensors.

Der Ausgabekontext der Lüftungsadaption wurde durch einen zusätzlichen Interpreter geschützt, der im Falle eines raschen Absinkens der Innentemperatur nach Beginn der Fensteröffnung die Aktion rückgängig machen konnte, bevor sich die Raumtemperatur insgesamt zu stark absenkte.

Ist es trotz der Berücksichtigung von Sonderfällen (im obigen Beispiel die Ausnutzung eines nahe am Fenster platzierten Innentemperatursensors) nicht möglich, die Mess- und Erkennungsgenauigkeit zu verbessern, muss in der Regel für die Kalibrierung auf zusätzliche Sensoren in Form von direkten Benutzerinteraktionen zurückgegriffen werden.

So konnte das in 2.4.3 beschriebene SUB der irrtümlichen Annahme eines Urlaubs bei längerem Ausbleiben an Wochenenden durch einen Taster kalibriert werden, der dem System ein bewusstes kurzfristiges Verlassen der Wohnung signalisierte. Natürlich stieg damit die Gefahr, dass auch in Urlaubssituationen aus Gewohnheit der Taster betätigt wurde. Die Urlaubserkennung wurde in solchen Fällen trotzdem aktiviert, allerdings erst nach zwei Tagen, wodurch Kurzurlaube möglicherweise nicht richtig erkannt wurden. Abhilfe schaffte eine Änderung des Bedienkonzeptes. Einmaliges Drücken des Tasters wurde als normales Verlassen interpretiert, zweimaliges Drücken als längeres Ausbleiben und drei- oder mehrmaliges Drücken als Urlaub. Obwohl die zusätzlichen Informationen keine Verbesserung der Erkennungssituation in Fällen von Vergesslichkeit brachten, wurden die Benutzer zusammen mit akustischen Rückkopplungen darauf konditioniert, besser auf die richtige Bedienung zu achten.

Eine ähnliche Kalibrierungslösung wurde auch für das komplizierte Problem der Lichtsteuerung gefunden. Das Störungsproblem wurde dadurch gelöst, dass nach einer bestimmten Uhrzeit automatisch nur mehr eine Notbeleuchtung aktiviert wurde. Die Hauptbeleuchtung konnte dann nur noch über manuelle Taster aktiviert werden. Ähnlich wurde das Problem mit den unterschiedlichen Intensitäten gelöst. Durch die Automatik wurde lediglich eine automatische Grundbeleuchtung garantiert. Verlängerungen der Aktivierungszeiten oder höhere Intensitäten mussten manuell signalisiert werden, falls keine automatische Erkennung möglich war. In manchen Räumen wurden durch Kalibrierung auch so genannte Beleuchtungsprogramme hinzugefügt, die sich entweder mit bestimmten Ereignissen (Aktivierung des Fernsehers etc.) oder über Fernbedienungen abrufen ließen. Theoretisch wäre aber auch eine weitere Verknüpfung mit Sprachkommandos denkbar gewesen.

Semiautomatik

Kontextadaptionen mit einem gewissen Anteil an direkter Nutzerinteraktion, wie die gerade beschriebenen, tragen ein erhöhtes Risiko für SUB-Phänomene in sich, die auf einer Unkenntnis des Benutzers über den notwendigen Grad manueller Einflussnahme beruhen. In besonderem Maße gilt das, wenn sich der Anteil direkter Interaktionen ebenfalls situationsabhängig verändert. Geeignete Kalibrierungen behandeln die SUB-Phänomene dieser Kategorie als ein Nebenläufigkeitsproblem mit einer zusätzlichen Absicherung einer Grundversorgung.

Als Beispiel dieser SUB-Kategorie war in 2.4.3 das Problem der semiautomatischen Lichtsteuerung genannt worden, das sich als Folge der Kalibrierung der Fehlinterpretationen ergeben hatte. In manchen Fällen wartete der Benutzer auf eine ausbleibende automatische Adaption, in anderen Fällen neutralisierten sich Benutzer und Systemautomatik gegenseitig.

Der erste Fall kann mit der im letzten Abschnitt bereits eingeführten Grundversorgung kalibriert werden. Der Benutzer erhält somit immer eine Grundreaktion und kann dann entscheiden, ob eventuell darüber hinaus bestehende Bedürfnisse durch direkte Interaktionen angestoßen werden. Kollisionen zwischen durch direkte Interaktion veranlassten Aktionen und automatischen Adaptionen des Systems lassen sich dagegen durch Hinzufügen von Synchronisationsinterpretern kalibrieren, welche die direkten Nutzerinteraktionen als eine zur Automatik parallele Kontextadaption (Taster als Sensor) auf eine gemeinsam genutzte Ressource (Aktuator der Lichtsteuerung) betrachten.

Generelle Ausnahmen

Alle in der letzten Kategorie genannten Fallbeispiele für SUB wurden im Prototyp durch die dritte Standardkalibrierung, also dem zeitweisen Deaktivieren einer bestimmten Adaption kalibriert, sofern sie nicht schon wie im Fall der Lichtsteuerung durch die Kalibrierungen anderer SUBs beseitigt waren. Der Grund dafür ist einfach nachvollziehbar. Alle verbleibenden SUB können in der Regel durch die vorhandenen Benutzerschnittstellen für Kalibrierungsausgaben oder die Online Hilfe im Voraus erkannt werden. Die geschilderten Situationen treten aber zu selten auf, um eine spezielle Kalibrierung dafür zu entwickeln.

8.2 Zusammenfassende praktische Bewertung

Gerade die im letzten Abschnitt beschriebenen Untersuchungen der Fallstudie haben gezeigt, dass das in dieser Arbeit beschriebene Konzept der kalibrierbaren Kontextadaption sowohl für die Dokumentation, den Entwurf als auch die Realisierung von Adaption sehr gut geeignet sind. Man kann auch sagen, dass K-Modelle ein probates Mittel darstellen, den Umgang mit technischer Softwareflexibilität (Webservices, mobile Komponenten etc.) und daraus resultierender Dynamik stark zu vereinfachen. Diese bessere Beherrschung der Möglichkeiten ist es letztendlich, die eine wirksame Bekämpfung des SUB-Phänomens erlaubt.

Zunächst ein wenig überraschend dagegen war die hohe relative Performanz des Prototyps aus der Telekommunikationsfallstudie [95] im Vergleich zu anderen konventionellen Adaptionsmodellen. Bei genauerer Überlegung ist die Erklärung aber sehr einfach. Viele Middlewareansätze für Adaption setzen bereits eine sehr hohe Softwareflexibilität als Basis für Änderungsmöglichkeiten am Kernsystem voraus. Im Fall einer tatsächlichen Rekonfiguration fällt daher die vorausgehende Anwendung der gleichen Flexibilitäten auch auf die Adaptionsschicht nur wenig ins Gewicht. Anders sieht das in der Theorie aber bei der Hintergrundbelastung aus. Da es sich bei den meisten Anwendungsfällen für Kontextadaption aber auch um verteilte Systeme handelt, die vor allem durch die Kommunikation ausgebremst werden, fällt der Mehraufwand für eine kalibrierbare Kontextadaption nicht ins Gewicht. Ganz im Gegenteil erlaubt die höhere Anpassbarkeit der Adaption selbst sehr starke Optimierungen bei gerade nicht benötigter Funktionalität. Konventionelle Adaptionsmodelle und deren Realisierung bestehen dagegen immer aus einer Superposition aller benötigten Features. Dadurch ist zu erklären, dass kalibrierbare Verfahren in normalen Anwendungsszenarien bis zu vier mal schneller als herkömmliche Adaptionsverfahren sind. Dieses Verhältnis wird natürlich schlechter, je weniger Änderungsmöglichkeiten ein Mechanismus am Kernsystem bereitstellt.

8.3 Praxisrelevanz

Im Gegensatz zu den technischen Vorzügen ist die Diskussion der tatsächlichen Praxisrelevanz wesentlich schwieriger. Obwohl mit dem SUB ein wichtiger Stolperstein auf dem Weg zu praktischer Anwendbarkeit beseitigt ist, gibt es noch mindestens zwei wichtige Problemstellen zu lösen.

Zum einen ist das die sich in den Fallstudien immer wieder zeigende mangelhafte Zuverlässigkeit der Hardware Komponenten. Das Problem ist dabei einfache Stochastik. Auch wenn die Ausfallsicherheit und Lebensdauer einer einzelnen Hardware Komponente durchaus zufrieden stellend ist, multiplizieren sich deren Ausfallwahrscheinlichkeiten aber in großen Systemen, die aus einer Vielzahl solcher Komponenten bestehen. Selbst in vergleichsweise einfachen ubiquitären Systemen wie dem der Fallstudie musste in mindestens monatlichem Rhythmus irgendeines der zahlreichen technischen Bauteile wegen Defekt ersetzt oder zumindest gewartet werden (Batteriewechsel etc.). Bevor solche Systeme reif für den Massenmarkt sind, muss hier also noch an der Wartungsfreundlichkeit der technischen Komponenten gearbeitet werden.

Ein weiteres Problem ist die in letzter Zeit stark zunehmende Diskussion über Datensicherheit und Datenschutz. Gerade die Kontextadaption ist hier besonders gefährdet, weil hier in großem Umfang auch sehr persönliche Daten über die Lebensumstände des Benutzers gesammelt werden.

8.4 Datenschutz

Neben einem nutzerfreundlichen und transparenten Umgang mit kontextadaptiven Systemen ist der Schutz und die Wahrung von Privatsphäre und die informationelle Selbstbestimmung der zweite wichtige Schlüssel für eine Nutzerakzeptanz neuer Technologien.

Smarte Gegenstände, kontextabhängige Anwendungen und ortsabhängige Dienste haben die prinzipielle Fähigkeit, Daten über den Aufenthaltsort, Vorlieben und Transaktionen der Nutzer zu sammeln und weitergeben. Mit der ubiquitären Erfassung von Nutzerprofilen läuft der Grundsatz der Zweckbindung der bisherigen Datenschutzgesetze, wonach die Erhebung und Verarbeitung Personendaten nur zu dem angegebenen Zweck zulässig sind, aber ins Leere. Das Anlegen personenbezogener Datensammlungen droht damit vom Ausnahme- zum Regelfall zu werden [64]. Die langfristig erkennbar werdenden Strategien verschiedener Industriezweige in puncto digitaler Rechteverwaltung (DRM[2]), Handel mit personenbezogenen Daten und von außen fremd verwalteter Homegateways weisen leider in eine völlig falsche Richtung. Anstelle von informationeller Selbstbestimmung und Transparenz treten eine informationelle Enteignung und Entmündigung des Kunden. Da-

[2] Digital Rights Management

gegen auf eine Korrektur dieses Irrweges durch staatliche Institutionen zu setzen bleibt auch in Zukunft zumindest tendenziell ein risikoreiches Unterfangen, solange allen ernstes Gesetzesvorlagen diskutiert werden, Kaminkehrer und Hausmeister im Dienste der Terror- und Verbrechensbekämpfung einzusetzen[65]. Die Begehrlichkeiten die eine Verfügbarkeit von detaillierten Daten über Bedürfnisse, Ess- Schlaf- und Unterhaltungsgewohnheiten oder Informationen mit noch weiter reichendem Missbrauchspotenzial womöglich noch zentral auf dem Firmenrechner eines Serviceproviders gespeichert, wecken könnten sind nur schwer einzuschätzen.

Auch wenn der einfache Mann von der Straße die Folgen zu Anfang nicht einschätzen können wird, wird sich das spätestens dann ändern, wenn die Auswirkungen greifbar werden. Viren, die statt einzelner Computer komplette Haushalte lahm legen oder die Nachbarn über zweifelhafte Ess- oder Freizeitgewohnheiten informieren sind nur eine der denkbaren Folgen unkontrolliert zentral gespeicherter oder mit Überwachungsschnittstellen ausgestatteter Kontextinformationen.

8.4.1 Transparenz und Selbstbestimmung als Datenschutz

Einen Ausweg aus dieser Situation bietet letztendlich nur eine völlige Transparenz und informationelle Selbstbestimmung des Benutzers ubiquitärer Systeme. Zum einen fördert die Kenntnis der gesammelten Daten die Sensibilität des Benutzers, zum anderen schafft die vollständige Kontrollmöglichkeit das notwendige Vertrauen. Die Kontrollmöglichkeiten sollten dabei:

- Zugriff auf die Daten nur Berechtigten und nur zeitlich begrenzt erlauben,
- dabei die Weitergabe an Dritte verhindern
- und im Falle von Verstößen eine rechtliche Handhabe gegen den Verursacher bieten.

8.4.2 Urheberrecht für Kontextinformationen

Für die Durchsetzung der informationellen Selbstbestimmungsrechte bietet sich die Chance, das in jüngster Zeit stark verschärfte Urheberrecht Ausnahmsweise einmal zum Vorteil der Endverbraucher und im Dienste des Datenschutzes einzusetzen. Möglich ist das, da das Urheberrecht auch ausdrücklich auf Datenbanken anwendbar ist. Die gleichen technischen Maßnahmen und gesetzlichen Regelungen, die einen Schutz von digitaler Musik, Texten oder Filmen gewährleisten sollen, ließen sich auch zum Schutz von Kontextinformationen einsetzen.

So könnten DRM Container beispielsweise einen zeitlich und gerätespezifisch begrenzten Zugang zu den Informationen sicherstellen. Ebenso könnten digitale Wasserzeichen im Falle einer Umgehung der Schutzmaßnahmen eine Rückverfolgung auf den Täter erlauben. Notwendig dafür ist lediglich, dass sämtliche Kontextinformationen auf einem vom Benutzer kontrollierten System entstehen. Dies ist immer dann leicht möglich, wenn die zentrale Hardwareplattform eines ubiquitären Systems sich im Besitz des Benutzers befindet (Smart Homes und PDA-basierte Agentensysteme etc.). Schwieriger wird die Lage dann, wenn diese physikalische Kontrolle nicht möglich ist, etwa im Falle von öffentlichen Smartspaces. Schwierig dürfte es auch sein, die Hersteller dazu zu bewegen, entsprechende Produkte auch anzubieten.

Hier wäre eine gesetzliche Regelung notwendig, die das Sammeln personenbezogener Daten nur in Form von durch den Betroffenen digital signierter Informationen erlaubt. Damit läge das alleinige Monopol der Erzeugung (oder Erzeugung im Auftrag) personenbezogener Daten bei den Betroffenen selbst und die Daten wären damit durch das Urheberrecht vor Missbrauch durch empfindliche Strafandrohungen, besonders im Fall von kommerzieller Verwertung geschützt.

8.5 Ausblick

Denkbare weiterführende Arbeiten im Bereich der kalibrierbaren Kontextadaption betreffen vor allem ihre Verwendung in einem vollintegrierten Entwurf. Vollintegrierter Entwurf bedeutet, dass ein Gesamtsystem aus einem kleinen allgemeinen Kern (seed) heraus durch eine Folge von Adaptionen konstruiert wird. Die Kontextadaption ist dabei die Abbildung des gesamten Entwurfsprozesses von der Anforderungsanalyse bis zur Implementierung inklusive aller Entscheidungsschritte und verworfener Alternativen.

Genau genommen war die Kalibrierbarkeit der K-Modelle lediglich ein Sonderfall eines solchen vollintegrierten Entwurfsprozesses, der es dem Benutzer erlaubt, außerhalb des originären Entwurfsprozesses nachträglich bestimmte Änderungen am System vorzunehmen.

Zeitraum und Umfang dieser Modifikationsmöglichkeiten lassen sich aber schrittweise ausdehnen. Zunächst nur auf die Einbindung von Updateprozessen durch den Hersteller. Die spannende Fragestellung dabei ist, ob es in weiteren Schritten gelingt, dies auch auf den Entwurfsprozess für vollständige Neuentwicklungen auszudehnen. In einem solchen Fall würde der Lebenszyklus eines Systems mit dem Start der Entwicklungsphase beginnen und der Kontext würde zu einer Realisierung eines integrierten Produktmodells. Ent-

wurfsentscheidungen wären als Kontextadaption abgebildet und könnten im Falle einer Fehlentscheidung rückgängig gemacht werden. Auf diese Weise wäre endlich eine wirklich enge Kopplung zwischen allen Phasen eines Softwareentwurfes erreicht und eine jederzeitige Nachverfolgbarkeit und Bewertbarkeit einzelner Entwicklungsschritte gegeben.

Detailliertere Untersuchungen zu dieser Thematik werden aber wohl erst dann aktuell, wenn die Anzahl von Anwendungen mit kalibrierbarer Kontextadaption zunimmt und sich die Frage nach einem über die Kalibrierung durch den Benutzer hinausgehenden Wartungsprozess dieser Systeme bei den Kunden vor Ort stellt.

Anhang

A1 Die 2x2 Dimensionen der Ubiquität

Validierung der situativen Definition von Ubiquität durch Einordnung existierender Definitionen..

Für eine Validierung der Sinnhaftigkeit der 1.1 hergeleiteten Definition von Ubiquitous Computing werden in diesem Abschnitt zunächst geeignete Dimensionen der Ubiquität definiert. In den dadurch aufgespannten Eigenschaftsraum können die vielen oft phänomenartig aufgemachten oder stark szenariobasierten Forschungsprojekte und Einzelaspekte der ubiquitären Computernutzung eingeordnet werden. Dies zeigt im ersten Schritt, dass durch die Definition auch wirklich ein geeigneter Überbegriff für die sehr breit gefächerte tatsächliche Verwendung der Bezeichnung Ubiquitous Computing gegeben ist. Im zweiten Schritt lassen sich auch verwandte und grundlegende Aspekte von Computernutzung wie beispielsweise das Mobile- und Wearable Computing in einen solchen Eigenschaftsraum einordnen. Aus der Nichtexistenz von Überschneidungen mit existierenden Nutzungsformen in einem bestimmten Teilbereich des Eigenschaftsraumes kann als Validierung wiederum die Nutzbarkeit in unterschiedlichen Situationen als ein kennzeichnendes Merkmal abgelesen werden, durch das sich ubiquitäre von beispielsweise den mobilen Anwendungen abgrenzen lassen.

Technische Dimensionen sind für eine Einteilung ungeeignet.

Die Schwierigkeit bei der Angabe eines für die Validierung geeigneten Eigenschaftsraumes ist natürlich die Wahl der richtigen Einteilungskriterien. Als eine Möglichkeit der Einteilung bieten sich die wichtigsten technischen und methodischen Herausforderungen an, die ubiquitäre Systeme von bisherigen Anwendungen unterscheiden. In [31] werden für das Ubiquitous Computing beispielsweise die zehn Dimensionen *Bedeutsamkeit*, *Welt-Modellierung*, *Benutzer-Modellierung*, *Verteilung*, *Erreichbarkeit*, *Erweiterbarkeit*, *Heterogenität*, *Automatisierung*, *Verwendbarkeit* und *Ubiquität* vorgeschlagen. *Benutzer-Modellierung* bedeutet dabei beispielsweise eine Charakterisierung danach, ob die zu einer Person gehörenden Daten zentral in einem von überall aus erreichbaren Repository gelagert, oder mit der jeweiligen Person herumgetragen werden. Die einzelnen Dimensionen unterscheiden sich aber stark im Detaillierungsgrad. Teilweise sind einige der Dimensionen lediglich eine Kombination aus anderen. Beispielsweise umfasst die Dimension der *Bedeutsamkeit* von Interaktionen auch die Dimensionen *Automatisierung*, *Welt-* und *Benutzermodellierung*. Die in [31] getroffene Einteilung ist auch deshalb ungeeignet, weil sie ein ideales ubiquitäres System anhand des Erfüllungsgrades fester technischer Herausforderungen und nicht wie in der Definition von Ubiquitous Computing gefordert, anhand der Erreichung eines Ziels charakterisiert. Das ist ein Nachteil, weil der technisch wertende Faktor der Einordnung nicht berücksichtigt, dass ein einziges flaches ubiquitäres System weder realistisch noch erwünscht ist. Vielmehr wird es zumindest in Teilsysteme (sog. ubiquitäre Inseln) unter-

teilt sein, für die nicht alle technischen Herausforderungen in gleichem Maße ausgeprägt sind und deshalb bezüglich der zuletzt genannten Dimensionen geringerwertige Lösungen die weniger aufwändigen und damit besseren sind.

Hauptdimensionen Verfügbarkeit und Nutzbarkeit.

Eine Einteilung anhand von wichtigen Eigenschaften ohne Berücksichtigung ihrer technischen Realisierung erscheint daher als die vernünftigere Alternative. In der Herleitung der Definition wurde dazu bereits argumentiert, dass Ubiquität hauptsächlich bezüglich zweier Eigenschaften wachsen kann, in der Verteilung und in der Interaktion. Die Verteilung lässt sich genauer charakterisieren als die Art, wie ubiquitäre Systeme in Form von Funktionen (Berechnungen) und Daten im realen Raum verteilt und somit dem Benutzer verfügbar gemacht werden. Die Interaktion beschreibt die Art, wie das System dann für den Anwender nutzbar wird. Mithilfe dieser zwei Dimensionen lässt sich ein ubiquitäres System bereits grob charakterisieren und abgrenzen, wenn man als Maße die durch die Art der Verteilung erreichte Überall-Verfügbarkeit und die durch den Interaktionsmechanismus erreichten Automatisierungsgrad bewertet. Die Gesamtubiquität lässt sich als Überall-Nutzbarkeit und damit als Kombination aus Verfügbarkeit und Aufwand bei der Interaktion ausdrücken. Ein System, das zwar überall verfügbar, aber nur umständlich oder langwierig zu bedienen ist, ist damit insgesamt seltener nutzbar als ein teilweise automatisiertes System, das aber nicht überall verfügbar ist.

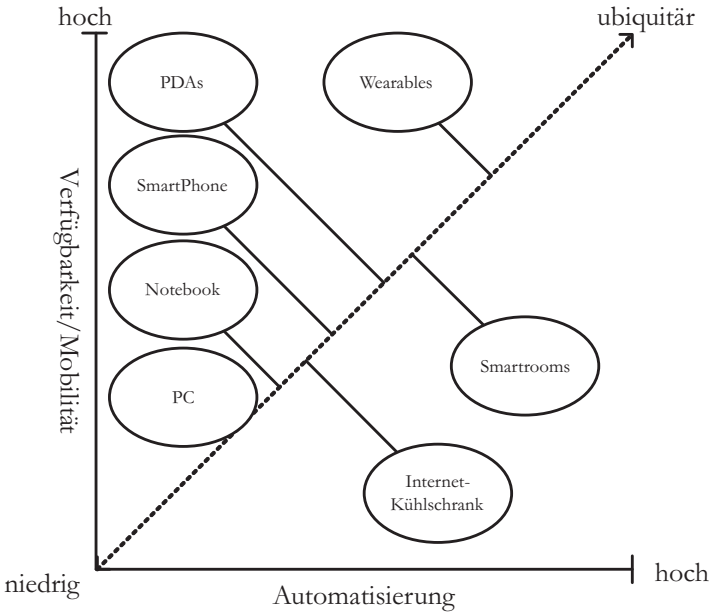


Abb. 82: Einteilung ubiquitärer Technologien

Abb. 82 zeigt eine solche erste Einteilung ubiquitärer Produkte und Anwendungsgebiete. Tragbare nicht über ein WLAN verteilte Systeme wie PDAs und Wearables bieten dabei potenziell die höchste Verfügbarkeit, da sie nicht auf die Erreichbarkeit anderer Komponenten angewiesen sind, die mögli-

cherweise gewissen technischen Restriktionen (Reichweite, Bandbreite etc.) unterliegen. Dafür sind sie durch die Portabilität in ihrer Leistungsfähigkeit beschränkt, was bestimmte aufwändige Formen der Interaktion und Automatisierung normalerweise ausschließt. Smartrooms unterliegen natürlich gewissen Verfügbarkeitsbeschränkungen, vereinen aber die Vorteile einzelner Geräte (z.B. Internet Kühlschrank) und können aufgrund der höheren möglichen Gesamtrechenleistung auch aufwändige Interaktionsmechanismen nutzen. Das untere Ende der Ubiquität markiert die Nutzung von PCs, die nur an wenigen stationären Orten genutzt werden können. Dennoch sind PCs vergleichsweise sehr leistungsfähig und flexibel einsetzbar und es ist möglich, auch entfernt auf seine Daten und Funktionen zuzugreifen, entweder über Internet, oder einen transportablen Datenträger.

Unterteilung der Verfügbarkeit nach Funktion und Zugang.

Für eine differenziertere Einteilung der verschiedenen Produkte, Szenarien, Technologien und Forschungsbereiche, die unter dem Begriff Ubiquitous Computing zusammengefasst werden, lässt sich das in Abb. 82 aufgespannte Gebiet noch in jeweils zwei weitere Dimensionen mit deutlicheren Wertungskriterien unterteilen.

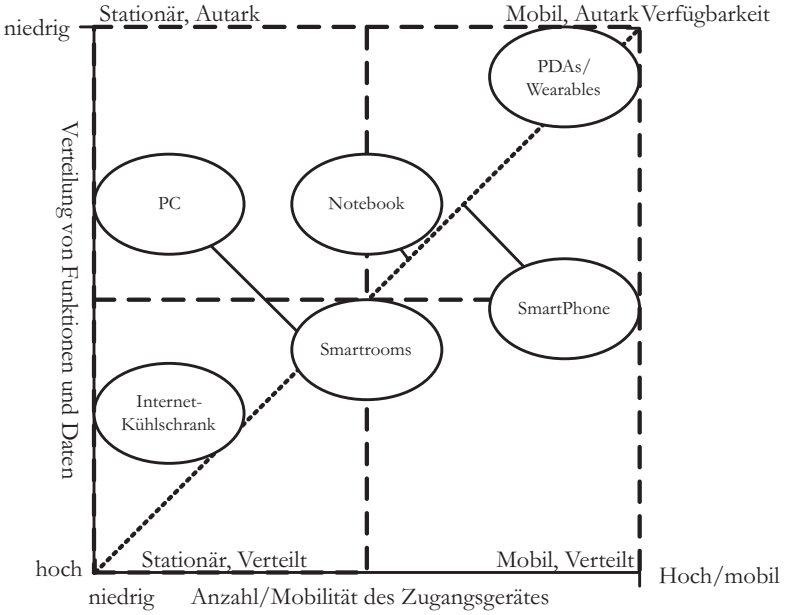


Abb. 83: Dimensionen der Verfügbarkeit

Für die Dimension der Verfügbarkeit, die über die Verteilung gemessen wird, geling das, indem man genauer zwischen der Verteilung von Daten und Funktionen auf der einen und der Verteilung der Zugangsgeräte auf der anderen Seite unterscheidet. Eine Überall-Verfügbarkeit kann nämlich abstrahiert gesehen über vier Grundmöglichkeiten erreicht werden. Auf der mobilen Seite werden die Grenzen durch den Zugang über ein omnipotentes mobiles Zu-

gangsgerät gezogen, das alle Daten und Funktionen enthält, bzw. durch ein relativ einfaches tragbares Terminal, das den Zugriff auf die Vernetzung aller Daten und Funktionen in einer Art von globalem Netzwerk (Evernet) erlaubt (Abb. 83). Auf der stationären Seite sind die Grenzen der Einteilung einmal der Zugriff auf das Evernet unspezifisch über eine große Zahl in der Umwelt verteilter Gegenstände, bzw. durch die Nutzung stationärer omnipotenter Terminals in gewissen Einzugsräumen. Das Wertungsmerkmal der Verfügbarkeit lässt sich mit dieser Einteilung einerseits über die geografische Erreichbarkeit der Zugangsgeräte und andererseits über die Abhängigkeit von anderen Komponenten definieren. Mobile Zugangsgeräte haben die potenziell größte Verfügbarkeit gefolgt von in die Umwelt integrierten unspezifischen Zugangsgeräten abhängig von der Dichte der Verteilung. Je mächtiger das Endgerät, desto geringer ist die Abhängigkeit von anderen verteilten Geräten und desto unwahrscheinlicher sind temporäre Fluktuationen in der Leistungsfähigkeit und damit die Verfügbarkeit einer Anwendung. Bei der Einteilung ist zu beachten, dass als Maßstab jeweils immer eine generelle Verwendung herangezogen wird. Dadurch ist ein Smartroom global betrachtet nicht generell nahtlos verfügbar, obwohl natürlich innerhalb des Smartrooms in der Regel eine Überall-Verfügbarkeit gegeben ist.

Unterteilung der Nutzbarkeitsdimension in die Automatisierung von Interaktion und Integration.

Neben der reinen ubiquitären Verfügbarkeit ist aber auch die tatsächliche Nutzung entscheidend. Diese ist abhängig von der Art der Interaktion zwischen dem System und einem potenziellen Benutzer. Verlangt ein System die volle Aufmerksamkeit, so ist es beispielsweise wie das Fernsehen trotz Verfügbarkeit nicht nutzbar, wenn der Benutzer sich zur gleichen Zeit auch noch auf etwas anderes konzentrieren (etwa Auto fahren) muss. Wird über akustische Signale interagiert (z.B. Navigationssystem) können dagegen möglicherweise mehrere Tätigkeiten parallel ausgeführt werden. Indirekt betrifft das auch die Interaktion zwischen heterogenen Komponenten. Je geringer die Voraussetzungen an potenzielle Kooperationspartner (die Infrastruktur), desto mehr Möglichkeiten gibt es, eventuell störende technische oder physikalische Restriktionen automatisiert zu überwinden, oder neue Anwendungsfälle durch eine Kombination von Einzelanwendungen abzudecken, ohne den Benutzer zu involvieren. Ein Beispiel ist die Anpassung eines Notebooks an eine geänderte Infrastruktur, etwa in Form von Einstellungen neuer Netzwerkparameter an jedem neuen Ort.

Für die Unterdimension der reinen Benutzerinteraktion bietet sich natürlich zunächst eine Einordnung dahingehend an, inwieweit die Interaktion dem Ideal von Mark Weiser respektive der *Calm Technology* oder einer anderen Automatisierung nahe kommt. Der gleiche Umstand lässt sich aber auch leichter messbar durch die Häufigkeit/Dauer expliziter Interaktionen darstellen. Die Grenzen werden markiert durch kommandozeilenorientierte Schnittstellen

auf der einen und vollautomatischen Systemen ohne oder mit als natürliche Verstärkung empfundenen Ausgaben wie bei Bremsassistenten auf der anderen Seite (siehe Abb. 84).

Die Interaktion zwischen technischen Teilsystemen lässt sich dagegen einfach charakterisieren durch die automatische Heterogenität/Spontanität der miteinander kommunizierenden Teilsysteme.

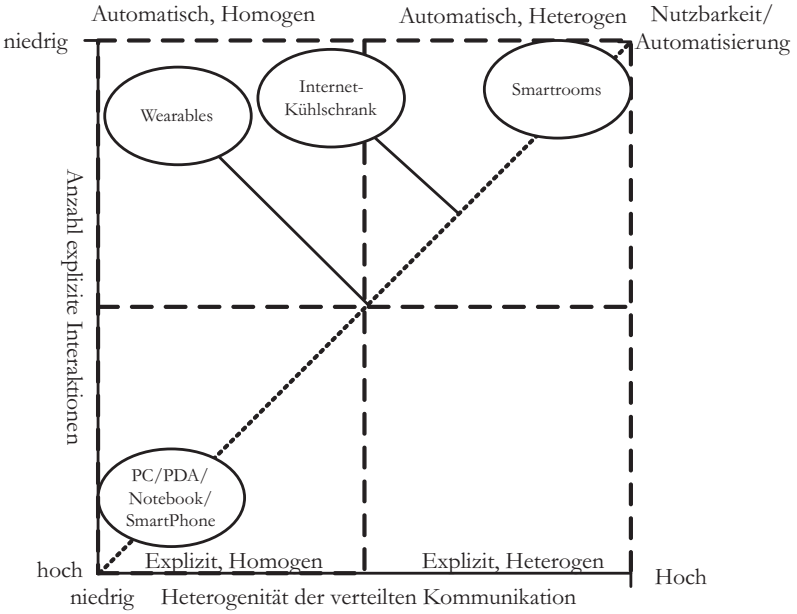


Abb. 84 : Dimensionen der Nutzbarkeit

Damit erfolgt die Einteilung also anhand der Flexibilität der Kommunikationsschnittstelle und danach, inwieweit Syntax und Semantik von Nachrichten zwischen Partnern zur Entwicklungszeit festgelegt wurden und bei der Kommunikation implizit vorausgesetzt, oder explizit zwischen den Partnern zur Laufzeit verhandelt werden können.

Ubiquitäre Systeme lassen sich insgesamt also nach den Dimensionen Verteilung von Funktionen und Daten, Verteilung der Zugangsgeräte, Automatisierung der Benutzerschnittstelle und Automatisierung (ad hoc Fähigkeit) der Kommunikationsverbindung zwischen heterogenen Komponenten einordnen. Die ersten beiden Dimensionen beschreiben die Verfügbarkeit, die letzten beiden die Nutzbarkeit einer Anwendung in einer bestimmten Situation (Ort, Zeit etc.) des Benutzers. Alle vier Dimensionen zusammengenommen erhält man dadurch aus der situativen Definition des Begriffes Ubiquitous Computing einen Eigenschaftsraum, in den sich die wichtigsten Begriffsdefinitionen aus der Literatur und Praxis einordnen lassen.

A1.1 Einordnung bekannter Aspekte des Ubiquitous Computing

Durch die im letzten Abschnitt beschriebenen vier Dimensionen ist ein Eigenschaftsraum gegeben, in dem sich sowohl verschiedene prinzipielle Möglichkeiten der Nutzung mehrerer Computer (parallel, sequenziell) als auch einzelne Anwendungsszenarien und Realisierungstechnologien einordnen lassen. Da eine vollständige Beschreibung und Einordnung aller Aspekte ubiquitärer Computernutzung den Rahmen dieser Arbeit sprengen würden, finden sich im Folgenden lediglich einige wenige Beispiele stellvertretend für viele andere für diese Arbeit untersuchte Aspekte.

Mobile Computing

Diese Form der Computernutzung geht vor allem einher mit der Verbreitung von Mobiltelefonen und der damit verbundenen Verfügbarkeit weiträumiger drahtloser Vernetzung. Gleichzeitig existieren Konvergenzbestrebungen, Telefonie, Multimedia und Datendienste miteinander zu verschmelzen und dadurch neue Anwendungsfelder für den Mobilfunk zu erschließen. Andererseits ermöglicht die fortschreitende Miniaturisierung, auch leistungsfähige Computer in tragbarer Form zu entwickeln und ortsungebunden einzusetzen. Innerhalb des Mobile Computing gibt es wiederum weitere Unterteilungen, beispielsweise nach Art der verwendeten Hardware wie im Wearable Computing, oder der Art und Weise der Nutzung bzw. genutzten Anwendungen im Nomadic Computing oder den Location Based Services. Mobile Computing wird damit oft vergleichbar dem Ubiquitous Computing als Überbegriff verwendet und manchmal sogar mit diesem gleichgesetzt [41].

Das engere Forschungsgebiet Mobile Computing wird normalerweise mit der Mobilität von Hardware, Daten und Software in Computeranwendungen assoziiert. Theoretisch erfüllen auch in sich abgeschlossene Datenverarbeitungsgeräte wie (ältere) PDAs diese Bedingung, in der Regel beschäftig sich Mobile Computing aber mit tragbaren frei beweglichen Rechnern, die über ein drahtloses Kommunikationsmedium ständig oder drahtgebunden auch nur zeitweise mit anderen Computern verbunden sind.

Im Bezug auf die in dieser Arbeit vertretene Definition von Ubiquitous Computing ist Mobile Computing lediglich eine von zwei technischen Grundmöglichkeiten, Computer in die Umwelt zu bringen, um dadurch eine Überall-Verfügbarkeit zu erreichen. Dass ein Computer mobil ist, bedeutet aber noch lange nicht seine vermehrte oder sogar Überall-Nutzung. In Form von ad hoc Kommunikation zwischen mobilen Geräten und einer wechselnden, zum Entwurfszeitpunkt möglicherweise unbekannten Infrastruktur beschäftigt sich das Mobile Computing bereits in Richtung einer Situationsabhängig-

keit, allerdings steht dabei der hardwaretechnische und damit der Aspekt der Verfügbarkeit im Vordergrund (siehe Abb. 85).

Wearable Computing

Wearable Computing entstand als ein Spin-off des Mobile Computing und bezeichnet Computer, die in Form von Brillen, Handschuhen, Kleidungsstücken bis hin zu Implantaten getragen werden. Auf der Suche nach einer geeigneten Definition wird man öfter auf eine Definition [43] des MIT Medialab verwiesen. Diese Definition beschränkt sich jedoch auf eine Sammlung hinreichend ungenauer Beispiele, so dass der Unterschied zu anderen tragbaren Computern nicht erkennbar ist. Es existieren aber geeignetere Definitionen [42], welche die Unterscheidungsmerkmale besser herausarbeiten. Darin wird Wearable Computing als eine neue Form der Mensch-Computer Schnittstelle propagiert, die auf Geräten basiert, die ständig in Benutzungsbereitschaft und verfügbar sind. Oft wird neben der Portabilität und dem kontinuierlichen Betrieb noch das Merkmal einer freihändigen Bedienung gefordert. Dadurch grenzt sich das Wearable Computing selbst von einfachen tragbaren Geräten wie PDAs, Notebooks und Handhelds ab. Die jederzeitige Verfügbarkeit dabei soll über eine längerfristige Gewöhnung an eine konsistente Schnittstelle zu einer Synergie zwischen Mensch und Computer (Cyborg) führen.

Als Forschungsgebiet beschäftigt sich das Wearable Computing mit ähnlichen Fragestellungen der Mobilität wie das Mobile Computing, jedoch tritt die technische Interaktion mit wechselnden heterogenen Infrastrukturen in den Hintergrund. Meist trägt der Benutzer eine aufeinander abgestimmte homogene Ausrüstung, die als Body Area Network kommuniziert und die meisten Funktionen aus der Sicht des Benutzers autark erbringt und höchstens über ein drahtloses Netz auf nicht mehr transportable oder aktualisierte Daten zugreift. Dafür rückt der Aspekt der Interaktion mit dem Benutzer stärker in den Vordergrund, da das System permanent am oder sogar im Körper getragen wird und dadurch ein hohes Potenzial von Störungsmöglichkeiten entfaltet. Die Fragestellungen diesbezüglich formulieren dadurch ein Teilgebiet des Ubiquitous Computing, das Heterogenität so weit wie möglich vermeidet. Konservativere Benutzernaturen werden sich aber wohl lieber auf eine etwas heterogene Mischform aus tragbaren und in die Umgebung integrierten Computern mit möglicherweise weniger konsistenten Schnittstellen einlassen, als auf den Synergieeffekt einer Verschmelzung aus Mensch und Maschine zu hoffen.

Nomadic Computing

Mit wenigen Ausnahmen existieren auch für den Begriff *Nomadic Computing* nur wenig klare Definitionen. Eine Ausnahme davon ist [44]. Darin wird ein Benutzer beschrieben, der sich zwischen verschiedenen Infrastrukturen bewegt und die jeweils an einem Ort vorgefundene Infrastruktur dazu benutzt, sich mit einem Heimatsystem zu verbinden und dessen Dienste zu nutzen.

Ein wesentliches Unterscheidungsmerkmal zu den bisher betrachteten Aspekten des Ubiquitous Computing ist die Tatsache, dass im *Nomadic Computing* meist ein expliziter Verbindungsaufbau über die wechselnde Infrastruktur stattfindet und sich zudem während der Dauer der Verbindung der Aufenthaltsort des Benutzers nicht verändert. Das Nomadic Computing konzentriert sich also darauf, die gleiche Anwendung an jedem Ort in gleicher Weise transparent, d.h. ohne ortsspezifische Bedienungsschritte zur Verfügung zu stellen [44]. Ortsbasierte Dienste stellen davon das genaue Gegenteil dar, da deren Nutzung nur an einem bestimmten Ort möglich oder sinnvoll ist, wie etwa ein Stadtplan für die Stadt, in der man sich gerade befindet.

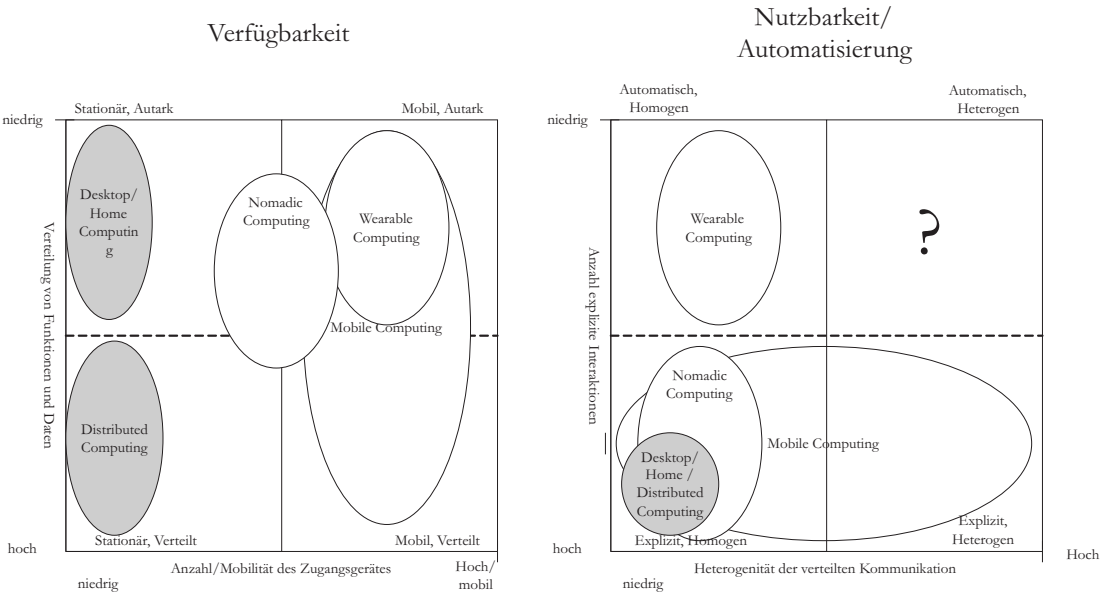


Abb. 85: Verschiedene Aspekte von Ubiquitous Computing

Abbildung 85 zeigt das Ergebnis der Einordnung der beispielhaft betrachteten Aspekte des Ubiquitous Computing in die im letzten Abschnitt definierte Einordnung. Zum besseren Vergleich enthält Abbildung 85 auch eine Einordnung der Grenzen des Ubiquitous Computing in Form von isolierten Einzelplatz-(Desktop) und verteilten Anwendungen (Distributed Computing). Innerhalb des durch die Dimensionen des Ubiquitous Computings aufge-

spannten Raumes tritt dadurch ein Sektor in den Vordergrund, der die Nutzbarkeit durch automatische Interaktion mit und zwischen heterogenen Softwarekomponenten besonders betont. Abbildung 85 zeigt dadurch, dass das Ubiquitous Computing zwar einen großen Bereich der Computernutzung abdeckt und dabei auch spezialisiertere Aspekte der Computernutzung (mobil, wearable etc.) integriert, das wesentliche Unterscheidungsmerkmal zwischen ubiquitären und sonstigen Formen mobiler, tragbarer oder stationärer Computernutzung aber in einer Betonung der Nutzbarkeit gegenüber der reinen technischen Verfügbarkeit liegt. Die Nutzbarkeit erstreckt sich dabei nicht nur auf die Interaktion zwischen Benutzer und Anwendung, sondern auch auf die Integration von heterogenen Einzelanwendungen in Form von Softwarekomponenten oder Diensten zu einer den jeweiligen Bedürfnissen des Benutzers entsprechenden Gesamtanwendung. Diese Beobachtung korrespondiert mit der in 1.1 abgeleiteten Behauptung, die Nutzbarkeit in unterschiedlichen Situationen sei das wesentliche Merkmal von Ubiquitous Computing.

A1.2 Höchste Ubiquität einer Anwendung?

Die gerade diskutierten Dimensionen der ubiquitären Computernutzung und das daraus identifizierte wesentliche Merkmal ubiquitärer Anwendungen werfen aber auch die Frage auf, ob Ubiquität sich auf eine messbare oder zumindest vergleichbare Eigenschaft zurückführen lässt und damit konsequenterweise eine am meisten ubiquitäre Anwendung existiert. Eine solche Eigenschaft kann durch die folgende Überlegung erkannt werden.

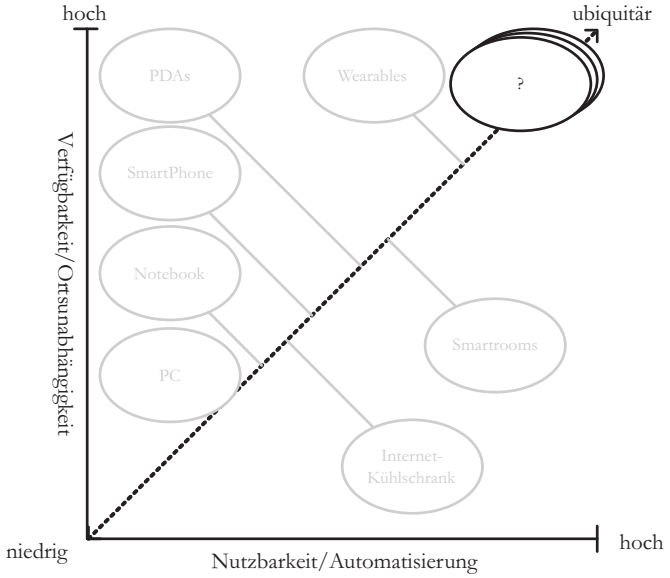


Abb. 86: Das Ideale Ubiquitäre System?

Bezüglich eines Maßes für die Ubiquität eines Systems ist die durch die Verfügbarkeitsdimension erreichte Ortsungebundenheit oder Überall-Verfügbarkeit notwendige Voraussetzung, die Dimensionen der Interaktion also die Nutzbarkeit sind aber entscheidend. Diese hängen nämlich davon ab, wie manuell aufwändig und exklusiv jeweils interagiert werden muss, egal ob es darum geht, viele Computer gleichzeitig oder sequenziell (in allen schwierigen Situationen) zu nutzen. Die Verfügbarkeit bildet dadurch eine obere Schranke für den Grad der Überall-Nutzbarkeit und damit eine notwendige Voraussetzung für ubiquitäre Anwendungen. Betrachtet man nun noch einmal die Einteilung der bekanntesten Produkte, Anwendungsgebiete und Technologien aus dem Bereich des Ubiquitous Computing in Abb. 82 und deren Ordnung bezüglich einer Ubiquität, die sich zu gleichen Teilen aus Verfügbarkeit und Nutzbarkeit zusammensetzt, so fällt auf, dass keines der betrachteten Beispiele eine maximale Ubiquität erreicht. Grund dafür sind praktische Restriktionen, denen die technische und wirtschaftliche Umsetzung in der Praxis unterworfen ist. So können Situationen eintreten, in denen ein Benutzer einen Smartroom verlassen muss, es aber gleichzeitig nicht vertretbar ist, den gesamten Weg zum nächsten Smartroom mit in der Umwelt integrierten Terminals zu überbrücken, um dem Benutzer Daten und Funktionalität auch von unterwegs an jedem Ort der Welt zugänglich zu machen. In diesem Fall wäre er mit einem mobilen Zugangsgerät besser bedient. Entfernt er sich noch weiter von der Zivilisation und damit verbundenen drahtlosen Netzwerken, wäre ein autarker Wearable Computer die beste Wahl. Trotzdem ist diese Lösung mit der besten Verfügbarkeit aus ebenso praktischen Restriktionen wie beispielsweise dem Energieverbrauch oder auch dem Bedienungskomfort nicht automatisch die mit der höchsten Nutzbarkeit und damit insgesamt der höchsten Ubiquität. Aus der anderen Richtung kann man nämlich argumentieren, dass sich eine Email-Nachricht zu Hause leichter per Spracheingabe diktieren lässt, als auf dem Nummernblock des Smartphones eingegeben zu werden und das Tragen einer Datenbrille im Bett, um per Augmented Reality die Position des Lichtschalters zu finden normalerweise unnötig und unbequem ist. Ein noch besseres ubiquitäres System ließe sich für die gegebenen Beispiele aber durch die Kombination aller genannten Beispiele erzielen (Abb. 86), wobei je nach Situation die jeweils ideale Technologie der Verfügbarkeit und die ideale Art der Anwendung beziehungsweise Automatisierung benutzt wird.

Als Schlussfolgerung kann man ziehen, dass die die Ubiquität bestimmtende Eigenschaft aus einer situationsabhängigen Realisierung einer situations- und benutzerspezifischen Anwendung und Interaktionsform besteht. Dadurch wird gegenüber bisherigen Nutzungsformen die Nutzung von Computern

und Computeranwendungen auf die maximal mögliche Anzahl von Situationen eines Benutzers ausgeweitet.

Mit diesem Ergebnis ist das für ubiquitäre Anwendungen wesentliche Merkmal der Überall-Nutzbarkeit wie behauptet, auf eine einzelne Eigenschaft der ubiquitären Anwendungen zurückgeführt. Wenn eine Anwendung die sie umgebende Situation als Kontext ihres Verhaltens und ihrer Realisierung berücksichtigt, kann sie sich damit theoretisch den unterschiedlichsten Situationen anpassen. Für einen Benutzer als Teil dieser Situation wird damit eine optimale Nutzbarkeit erzielt. Dadurch wird die Anwendung in möglichst vielen Situationen nutzbar und daher ubiquitär verwendbar.

A1.3 Abgrenzung zu anderen Fachgebieten

Obwohl die in dieser Arbeit verwendete Definition von Ubiquitous Computing einen sehr breiten Bereich umfasst, in den sich viele bekannte Aspekte und Techniken der Computernutzung einordnen lassen, kann zu einigen artverwandten Fachgebieten dennoch eine klare Abgrenzung definiert werden. Dies ist immer dann der Fall, wenn sich die verwandten Fachgebiete ebenfalls aus einer allgemeinen Zielsetzung heraus und weniger über die verwendeten Technologien definieren.

Robotik

Robotik beschäftigt sich mit der Realisierung möglichst flexibel einsetzbarer "intelligenter" Maschinen und ist damit auch für Software ein wichtiges Anwendungsfeld, in dem viele zentrale Gebiete der Informatik, wie beispielsweise Mustererkennung, Künstliche Intelligenz, Steuern und Regeln von Aktuatoren, Aufbau von Sensorsystemen und einiges mehr zusammenfließen und auch zum Teil interdisziplinär an Themenbereiche aus benachbarten Ingenieurwissenschaften, wie der Elektrotechnik und dem Maschinenbau anknüpfen. Die angewandten Methoden sind aber lediglich für einen Teilausschnitt ubiquitärer Anwendungen geeignet, da sie zwar dafür ausgelegt sind, in a priori unbekannten Umgebungen zu agieren, jedoch in der Regel eine statische Infrastruktur für das System selbst voraussetzen, das heißt eine bestimmte Plattform aus Ablaufumgebung, Sensorik und Hilfsfunktionen.

Im Unterschied dazu erlaubt das Ubiquitous Computing auch den Entwurf von Anwendungen, die nicht nur in vorher unbekannten Umgebungen operieren, sondern auch mit vorher unbekannten und veränderlichen Infrastrukturen zurecht kommen. Die Ausführung kann dabei über mehrere heterogene Systeme verteilt sein. Anders herum liegt die Betonung im Ubiquitous Computing auch nicht auf einer völligen Autonomie der Systeme, sondern

auf der Entlastung des Benutzers von einfachen, sich wiederholenden Integrations- und Anpassungstätigkeiten und einem daraus erzielten Mehrwert.

KI

Das Gebiet der Künstlichen Intelligenz (KI) dreht sich im Wesentlichen um die Frage, wie es Computern möglich ist, vergleichbar dem Menschen innerhalb einer komplexen Welt zu agieren, also sich wie ein Mensch zu verhalten oder wie ein Mensch zu denken. Grundlegendes Ziel ist, die Nützlichkeit von Computern durch das Studium von Prinzipien, welche die Intelligenz ermöglichen, zu verbessern [47]. Einzelne Prinzipien umfassen die Gebiete der Wissensrepräsentation, Schlussfolgerung und Problemlösung.

Hier zeigt sich natürlich eine Überschneidung mit dem Ubiquitous Computing bezüglich der grundlegenden Zielsetzung, die Nutzbarkeit von Anwendungen zu verbessern. Im Ubiquitous Computing kommt es aber nicht so sehr darauf an, eine einzelne Technik zu entwickeln, als vielmehr, situationsabhängig die gerade günstigste verfügbare Technik einzusetzen. Das muss mitnichten ein Verfahren sein, das versucht, Prinzipien der Intelligenz anzuwenden, auch wenn ein ubiquitäres System im Ergebnis davon möglicherweise intelligent wirken kann.

Das wesentliche Unterscheidungsmerkmal zwischen Systemen, die in der KI betrachtet werden und ubiquitären Systemen besteht darin, dass ein ubiquitäres System nicht nur sein beobachtbares Verhalten, sondern auch seine Struktur und Realisierung an die gegenwärtige Situation anpassen kann. So gesehen ist der Unterschied der gleiche wie im Bezug auf die Robotik. Die verschiedenen Verfahren zur Lösung von Entscheidungsproblemen und der Wissensrepräsentation in möglicherweise veränderlichen Umgebungen gehen im Bereich der KI in der Regel von einer statischen Kerninfrastruktur als Plattform für die eigene Ausführung aus. Ein ubiquitäres System kann das nicht notwendigerweise voraussetzen. Stellt eine Infrastruktur aber nur sehr geringe Rechenleistung zur Verfügung, dann kann ein komplexer intelligenter Mechanismus unter Umständen zu langsam reagieren, um das System nutzbar zu machen, obwohl es theoretisch verfügbar und intelligent ist. Ein einfacher regelbasierter Adaptionsmechanismus leistet in einer solchen Situation dagegen wesentlich bessere Dienste.

A2 Smarthome Szenario

Das im Folgenden beschriebene Szenario zeigt die Nutzung einer fiktiven Smarthome-Anwendung. Es diente im Wesentlichen der Ermittlung von Anforderungen an einen kalibrierbaren Adaptionsmechanismus. Die Prototypen der Fallstudie aus Kapitel 2 implementieren lediglich einen Teilausschnitt der dargestellten Funktionen.

Tag 1: Ein Arbeitstag

5:30 Uhr. Während die Bewohner A und B noch schlafen, schließt sich das Fenster im Schlafzimmer selbsttätig. Gleichzeitig beginnt die Heizung zu arbeiten und Schlafzimmer, Küche, und Bad auf 22° zu erwärmen.

5:45 Uhr. A und B müssen um 6 Uhr aufstehen. Es beginnt ein mehrstufiges Weckprogramm. Die Helligkeit im Raum wird stetig erhöht, dazu spielt sanfte Musik. In der Küche wird die Kaffeemaschine aktiviert.

5:55 Uhr. Die Jalousie im Schlafzimmer öffnet sich. In der Küche wird der Toaster gestartet.

6:00 Uhr. Ein lautes Wecksignal ertönt.

6:15 Uhr. Während die Bewohner noch beim Frühstück sitzen, wird die Fußbodenheizung im Bad aktiviert. Gleichzeitig die Raumtemperatur im Bad auf 24° erhöht.

6:20 Uhr. Bewohner A duscht, die Lüftung wird aktiviert, um die entstehende Feuchtigkeit abzutransportieren. Bewohner B öffnet den Kleiderschrank und wählt passende Bekleidung aus, während der lokale Wetterbericht für den heutigen Tag vorgelesen wird.

6:30 Uhr. Bewohner B ist im Bad und hört dabei die wichtigsten Schlagzeilen und Börseninformationen. Bewohner A kleidet sich ebenfalls an und hört dabei die lokale Wettervorhersage der Stadt, in die er heute reisen wird.

6:40 Uhr. Bewohner A verlässt das Haus.

6:45 Uhr. Bewohner B ist bereit zu gehen und kontrolliert noch mal sein Äußeres im Spiegel im Flur. Im Spiegel wird die Meldung angezeigt, dass der nächste Nahverkehrszug 10 Minuten Verspätung hat. Gleichzeitig wird eine Liste kleinerer anstehender Tätigkeiten eingeblendet. Bewohner B wählt das Blumengießen aus. Es werden die Pflanzen bildlich angezeigt, die ein Auffüllen des Wasserreservoirs benötigen.

6:50 Uhr. Ein Gong signalisiert, dass es Zeit ist, das Appartement zu verlassen. Bewohner B verlässt das Appartement durch die Eingangstüre. Ein Warnton ist zu hören. Eine akustische Meldung teilt Bewohner B mit, dass er seinen Schlüssel vergessen hat.

6:51 Uhr. Bewohner B verlässt die Wohnung. Hinter ihm wird die Türe verschlossen. Das Licht im Bad wird gelöscht, die Wasserventile verriegelt, die Alarmanlage aktiviert, der Server in der Küche abgeschaltet, ebenso die Warmhalteplatte der Kaffeemaschine. Die Temperatur wird in allen Räumen auf 18° gesenkt, im Schlafzimmer gelüftet.

9:20 Uhr. Der Paketdienst klingelt an der Tür. Er wird von einer Sprachausgabe darum gebeten, das Paket bei einem Nachbarn abzugeben.

10:43 Uhr. Das Telefon klingelt. Der Anruf wird automatisch auf das Smartphone von Bewohner A umgeleitet.

12:15 Uhr. Es klingelt an der Tür. Es wird ein Bild des Besuchers aufgezeichnet und er wird aufgefordert eine Nachricht zu hinterlassen, oder dreimal in schneller Folge zu klingeln, falls es sich um einen Notfall handelt. Der Besucher hinterlässt eine Nachricht. Seine Firma sei spezialisiert auf die Erneuerung von Türen und Fenstern. Er möchte fragen, ob Bedarf besteht und hinterlässt seine Telefonnummer.

17:30 Uhr. Bewohner B kommt mit Einkäufen beladen von der Arbeit zurück. Die Eingangstüre entriegelt sich automatisch bei Annäherung, das Licht im Flur schaltet sich ein, die Temperatur in allen Räumen außer dem Schlaf- und Arbeitszimmer wird erhöht. B bringt die Einkäufe direkt in die Küche. Das Licht in der Küche schaltet sich ebenfalls automatisch ein. B geht zurück in den Flur und legt die Kleidung ab. Die Sprachausgabe meldet zwei Besucher, darunter den Postboten, weiterhin drei neue Email-Nachrichten und blendet die Bilder der Besucher als Schaltflächen ein. Bewohner A holt kurz das abgegebene Paket. Zurückgekommen wird der Statusbericht fortgesetzt. Das System teilt mit, dass Bewohner A wahrscheinlich gegen 19.00 eintreffen wird und beginnt die erste Nachricht vorzulesen. Im Spiegel werden die Schaltflächen „Antworten“, „Nächste“, „Letzte“ und „Abbrechen“ eingeblendet.

B geht in die Küche, um die Einkäufe auszupacken. Die Sprachausgabe wird in der Küche fortgesetzt, die Schaltflächen erscheinen jetzt auf dem Bildschirm am Kühlschrank. Die zweite Nachricht ist etwas länger, deshalb wird nur eine Zusammenfassung gegeben. Beim Vorlesen der dritten Nachricht wählt B die Schaltfläche „Antworten“. Es werden weitere Optionen eingeblendet (Telefon, Text, Sprache). B entscheidet sich für einen Anruf. Der Angerufene ist aber nicht erreichbar und besitzt keinen Anrufbeantworter. Das

System fordert B auf, eine Nachricht zu sprechen, die dann als Nachricht verschickt wird.

17:40 Uhr. Das System beendet den Statusbericht mit der Meldung, dass es ansonsten keine weiteren Vorkommnisse gäbe. Danach startet sanfte Musik in gedämpfter Lautstärke, während B die Einkäufe verstaut.

17:45 Uhr. Bewohner B ist mit dem Einräumen der Lebensmittel fertig und begibt sich ins Wohnzimmer. Die Musikausgabe folgt dahin und das Licht wird angeschaltet, gleichzeitig die Jalousie heruntergelassen, um Einblicke von Gegenüber zu verhindern. Der Fernseher wird aktiviert und zeigt Auswahlschaltflächen für das Lesen der langen Email-Nachricht oder Unterhaltung. B entscheidet sich für die Email-Nachricht, die dann auf der LCD-Fläche der Fernbedienung angezeigt wird.

17:50 Uhr. B hat die Email-Nachricht fertig gelesen und entschieden, die Antwort auf später zu verschieben. B wählt nun die Funktionen Fernsehen und Nachrichten. Die Musikwiedergabe stoppt und das Licht wird auf 50% Intensität reduziert. Auf dem Bildschirm wird die letzte Aufzeichnung einer Nachrichtensendung von B's Lieblingssender gezeigt. Die Sportberichterstattung am Ende der Nachrichten wird automatisch übersprungen.

18:00 Uhr. B verlässt das Wohnzimmer und betritt die Küche. Auf dem Display der Küchenkonsole ist eine große Schaltfläche in den Vordergrund getreten, die vorschlägt, den Kochvorgang zu beginnen. Daneben befinden sich die zwei Vorgangsmodifikatoren „Heute nicht“ und „Später“. B wählt „Kochen“. In der Küche wird zusätzliches Licht über der Arbeitsplatte aktiviert. Im Wohnzimmer wird das Licht und der Fernseher deaktiviert. Auf dem Bildschirm der Küchenkonsole startet eine Kochanwendung, die mit gezielten Fragen über Kochzeit, Personen, und Vorräten ein Rezept auswählt. Die Parameter der Zeit und Personen werden schon mit 1h und 2 Personen vorbelegt. Der Kochvorgang ist eine bebilderte Schritt-für-Schritt Anleitung. Temperatur der Kochplatten und Kochzeit werden von der Applikation gesteuert.

18:55 Uhr. A kommt nach Hause und betritt das Apartment ebenfalls mithilfe der automatischen Türentriegelung. A hat tagsüber bereits alle Email-Nachrichten auf seinem Smartphone empfangen. A bevorzugt für seine Interaktion mit dem Heimsystem ein Sprachinterface. Die Sprachausgabe fragt nach, ob A nach dem Abendessen noch arbeiten will. A antwortet mit Ja. Die Anwendung trifft daraufhin die entsprechenden Vorbereitungen wie das Synchronisieren bestimmter Daten zwischen dem Firmen- und dem Heimnetz.

19:00 Uhr. A und B nehmen das Abendessen ein.

19:05 Uhr. Ein Anrufer versucht, B zu erreichen. Da A und B beim Essen nicht gestört werden wollen und der Anrufer den Notfallcode nicht kennt, klingelt keines der Telefone.

19:30 Uhr. A betritt das Wohnzimmer und nimmt auf dem Weg dahin das heute eingetroffene Paket mit. Der Computer aktiviert sich, ebenso das Licht. Das Paket enthält eine Fax/Drucker/Scanner Kombination. Sofort nach dem Anschluss des Gerätes erscheint auf dem Schreibtischterminal ein Dialog, der darauf hinweist, dass ein neues Gerät installiert wurde, und zur Nutzung bereit ist. Verbunden ist dies mit der Frage, ob A mit der Konfiguration der Kontexteigenschaften beginnen will. A bestätigt dies und bekommt als Erstes eine Einführung in die Situationseigenschaften des neuen Gerätes. Jede vordefinierte Situation wird mithilfe eines Beispiels erklärt. Das neue Gerät kennt folgende Situationen: Das Gerät ist an, aus, im Standby, jemand druckt, benutzt den Scanner, benutzt das Fax, eine Anzahl wartender Aufträge in der Warteschlange des Druckers, ein Fax wird empfangen, eine Anzahl Faxe wurde empfangen, Papier oder Toner gehen zur Neige/sind leer, Papierstau, Wartung wird benötigt, Defekt. Diese Situationseigenschaften lassen sich mit anderen Situationseigenschaften zu einem bestimmten Zustand des Kontextes definieren. Als Nächstes macht sich A mit den Diensten vertraut, die das neue Gerät exportiert. In diesem Fall fünf neue Dienste, Drucken, FaxVersender, FaxEmpfänger, Scanner und ein Gerätedienst. Zunächst konfiguriert A den Gerätedienst als Teil des Haussteuerungssystems. Damit werden automatisch bestimmte Rechte und Sicherheitseinstellungen gesetzt. Sodann überprüft A bestehende Situationsdefinitionen, ob diese durch die neuen Situationselemente genauer beschrieben werden können. Im vorliegenden Fall betrifft das lediglich die Situationen „Person am Arbeitsplatz“, falls jemand den Scanner benutzt sowie „Geräteaktivität“.

Das Heimsystem besitzt bereits Dienste um elektronische Faxe zu versenden und zu empfangen, deshalb deaktiviert A diese Dienste als Grundeinstellung, um zu verhindern, dass im Falle einer Nichterreichbarkeit des Internetdienstes zum Faxversenden der wesentlich teurere Dienst des neuen Gerätes verwendet wird. A möchte das Gerät lediglich nutzen, wenn er am Schreibtisch anwesend ist, deshalb erweitert er die bestehende Situation „Person am Arbeitsplatz“ um einen Vorgang, der das Gerät aktiviert und die Situation „Arbeitsplatz unbesetzt“ um eine Abschaltung des Gerätes 15 Minuten, nachdem die letzte Person den Raum verlassen hat. Andere Vorgänge, wie beispielsweise das Abschalten aller unwichtigen Geräte beim Verlassen des Appartements müssen nicht verändert werden, da diese für ganze Geräteklassen definiert worden sind. Die gerätespezifischen Situationen "Papier/Toner" verbindet A mit dem Nachbestellungsvorgang, der bereits günstigste Preise, Lieferzeiten und Urlaube berücksichtigt. Wartung und Defekt verbindet A anstatt der vor-

eingestellten Benachrichtigung des Kundendienstes mit einer Nachricht an ihn selbst. Die Benutzung des Scanners verbindet A mit dem bestehenden FaxVersender Dienst, so dass in Zukunft immer dann, wenn jemand ein Papier in den Scanner legt auf dem nächstgelegenen Terminal neben den Optionen „Speichern“ und „Email“ auch die Funktion „Fax“ angeboten wird.

19:45 Uhr. Das Licht flackert kurz und A erhält kurz darauf auf seinem Bildschirm eine Warnmeldung über einen kritischen Fehlerzustand angezeigt. Diese unterrichtet ihn über den Totalausfall des Hauptservers, der zur Folge hat, dass einige unwichtigere Dienste wie das Sprachinterface nicht mehr zur Verfügung stehen. Außerdem ist die Leistungsfähigkeit von As Arbeitsrechner stark eingeschränkt, bis der Schaden behoben ist, da jener einige Aufgaben des Servers mit übernehmen musste. Auf allen anderen Terminals im Appartement ist eine ähnliche Meldung zu lesen, zusammen mit einem Menüpunkt, die Fehlerdiagnose zu beginnen. A wählt den Menüpunkt und bekommt die Anweisung, sich zum Server zu begeben. A kommt dem nach und begibt sich in die Küche. Da er direkt beim Server das Terminal in der Küche schlecht ablesen kann, aktiviert er sein Smartphone, bekommt dort ebenfalls den Diagnosedienst angezeigt und wählt „Diagnose fortsetzen“. Einige Schritte später ist klar, dass die Sicherung im Netzteil des Servers angesprochen hat. A behebt den Fehler und startet den Server. Nach einiger Zeit erhält er die Meldung, dass der Fehler behoben wurde. Nachdem A noch ein paar weitere Fragen verneint und die Frage nach flackerndem Licht bejaht hat, erscheint der Hinweis, zur Vermeidung eines erneuten Fehlers dieser Art eine USV mit Überspannungsschutz zu installieren. A bestätigt die Übernahme eines entsprechenden Eintrages in die Aufgabenliste seines Kalenders.

Tag 2: Wochenende

7:22 Uhr. Tag 2 beginnt wie Tag 1 aber ohne Wecker und erst als A und B von selbst erwachen. Für das lange Wochenende ist eine Kurzreise geplant. Bei der Auswahl der Bekleidung werden die Bewohner von einer Wettervorhersage über 3 Tage am Ziel der Reise unterstützt. Als Transportmittel dient ein Auto. Kurz vor dem Verlassen der Wohnung werden sie über einen Stau auf der geplanten Route unterrichtet. Eine alternative Route wird in As Smartphone geladen, zusammen mit einem, der neuen Streckenführung angepassten Reiseführer mit einer Beschreibung der Sehenswürdigkeiten und Rastplätze.

9:00 Uhr. A und B verlassen das Apartment. Hinter ihnen wird die Türe verschlossen. Alle Lichter gelöscht, die Wasserventile verriegelt, die Alarmanlage

aktiviert, der Server in der Küche abgeschalten. Die Temperatur wird in allen Räumen auf 16° gesenkt.

14:05 Uhr. Es klingelt an der Tür. Es wird ein Bild des Besuchers aufgezeichnet und er wird aufgefordert eine Nachricht zu hinterlassen, oder dreimal in schneller Folge zu klingeln, falls es sich um einen Notfall handelt. Der Besucher signalisiert, dass es sich um einen Notfall handelt. Daraufhin wird eine direkte Verbindung zwischen der Türkamera und As Smartphone hergestellt. Der Besucher ist der Hausmeister der Anlage und erklärt A, dass im Apartment darunter ein feuchter Fleck an einer Wand zu sehen ist. Er möchte überprüfen, ob in As Apartment eine Wasserleitung geborsten ist. A erklärt, dass bei einem Wasserschaden vom Computer automatisch Alarm ausgelöst worden wäre. Der Hausmeister möchte dennoch die Wände mit den Wasserleitungen auf Feuchtigkeit überprüfen. A erklärt sich einverstanden, deaktiviert die Alarmanlage und öffnet über sein Smartphone die Wohnungstüre. Über die Kameras beobachtet er, wie der Hausmeister in Bad und Küche die Wände überprüft. Es kann kein Schaden festgestellt werden und der Hausmeister verlässt die Wohnung. A aktiviert wieder die Alarmanlage und schließt die Türe.

15:10 Uhr. B's Vorgesetzter ruft an. Er erhält eine freundliche Nachricht, dass B über das Wochenende verreist und nicht erreichbar ist.

16:00 Uhr. B's Mutter ruft an. Der Anruf wird auf B's Mobiltelefon umgeleitet. B's Mutter möchte morgen auf dem Weg zu einer Veranstaltung einen Kuchen vorbei bringen. B erklärt, dass niemand zu Hause ist, sie den Kuchen aber trotzdem vorbeibringen kann.

16:05 Uhr. B benutzt A's Smartphone, um eine einmalige Besucher-Zugangsberechtigung einzurichten.

Tag 3: Wochenende

10:05 Uhr. B's Mutter steht vor der Türe. Das System erkennt den Transponderschlüssel als zugangsberechtigt für den heutigen Tag zwischen 9 und 11 Uhr. Die Alarmanlage wird deaktiviert, die Türe geöffnet. Alle anderen Systeme bleiben deaktiviert. B's Mutter stellt den Kuchen in die Küche und verlässt das Apartment wieder.

22:30 Uhr. A und B kommen von ihrer Reise zurück. Das System schaltet in den normalen Betrieb.

A3 Abkürzungen

APD	**A**daptation **P**ropagation **D**elay
API	**A**dvanced **P**rogramming **I**nterface
APIP	**A**daptation **P**ropagation **I**nterruption **P**roblem
APP	**A**daptation **P**ropagation **P**roblem
DRM	**D**igital **R**ights **M**anagement
EIB	**E**uropean **I**nstallation **B**us
FP	**Frame-Problem**
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper **T**ext **M**arkup **L**anguage
HW	**H**ard**w**are
IT	**I**nformations**t**echnologie
KI	**K**ünstliche **I**ntelligenz
LAN	**L**ocal **A**rea **N**etwork
LCDP	**L**east **C**ommon **D**enominator **P**roblem
MIT	**M**assachusetts **I**nstitute of **T**echnology
MVC	**M**odel **V**iew **C**ontroller
OCL	**O**bject **C**onstraint **L**anguage

OM	**O**ptische **M**askierung
P2P	**P**eer-**to**-**P**eer
PC	**P**ersonal **C**omputer
PDA	**P**ersonal **D**igital **A**ssistant
PIM	**P**ersonal **I**nformation **M**anagement
RFID	**R**adio **F**requency **I**dentification
SDR	**S**oftware **D**efined **R**adio
SUB	**S**pontaneous **U**nexpected **B**ehavior
SW	**S**oft**w**are
TUM	**T**echnische **U**niversität **M**ünchen
UC	**U**biquitous **C**omputing
UML	**U**nified **M**odeling **L**anguage
URL	**U**niform **R**esource **L**ocator
USV	**U**nterbechungsfreie **S**trom**v**ersorgung
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WSDL	**W**eb **S**ervice **D**escription **L**anguage
XML	E**x**tensible **M**arkup **L**anguage
Y2K	**Y**ear **2000**

A4 Glossar

Adaption

Allgemein versteht man unter dem Begriff Adaptation oder verkürzt Adaption im alltäglichen Sprachgebrauch ein "Anpassungsvermögen an die Umwelt" [4], unter Anpassungsvermögen versteht man die "Fähigkeit, sich den gegebenen Verhältnissen anzupassen" [4] und anpassen bedeutet "sich nach jeweiligen Umständen zu richten" [4].

Der Begriff wird im Sprachgebrauch hauptsächlich verwendet zur Beschreibung biologischer oder medizinischer Vorgänge, bei denen sich ein Organismus oder Teil eines Organismus an die Umwelt anpasst [99].

Das Ziel einer Adaption ist immer eine Art von Verbesserung der von Umwelteinflüssen abhängigen Fähigkeiten des adaptierten Gegenstandes. Was bei biologischen Systemen die Überlebensfähigkeit ist, kann bei ubiquitären Systemen auf die Nutzbarkeit übertragen werden. Je besser es sich in jeder Situation anwenden lässt, desto häufiger wird es benutzt und "überlebt" da-durch.

Das während einer Adaption anzupassende System kann dazu seine innere Struktur, beobachtbare Form und sein Verhalten abhängig von Umweltbedingungen verändern. Für ubiquitäre Systeme bedeutet das eine Anpassung an eine Situation, die durch Informationen über die Umgebung erkannt werden kann und aus der sich bestimmte Anforderungen des Benutzers oder Betreibers an das System ableiten lassen. Beispiele dafür sind die automatische Aktivierung bestimmter Dienste, deren Qualität oder Ausführungsort.

Anwendung

Eine Anwendung ist ein System, das Software-Komponenten enthält, die zur Lösung einer Menge von inhaltlich zusammengehörigen Aufgaben notwendig sind [14][15].

Appropriate Computing

Computernutzung über dem jeweiligen Zweck angepasste explizite und impliziten Interaktionsmöglichkeiten (siehe auch [31]).

Augmented Reality

Der Begriff Augemented Reality geht zurück auf Ivan Sutherland, der die Idee hatte, die visuelle Wahrnehmung eines Benutzers mit virtuellen Informationen anzureichern. In [97] beschrieb er ein tragbares Sichtgerät, mit es

möglich war, die reale Welt mit der graphischen Darstellung einer virtuellen Welt zu überlagern.

Die meisten Definitionen von Augmented Reality nennen als wesentliche Charakteristika, dass es sich um eine interaktive dreidimensionale Kombination aus realer und virtueller Welt in Echtzeit handelt, z.B. in [98].

Calm Technology

Unbewusste respektive unterbewusste Nutzung von Computern. Die Ausgaben der Anwendung können dabei als Verstärkung der eigenen Fähigkeiten oder der natürlichen Eigenschaften von Gegenständen und Räumen empfunden werden (siehe auch [35]).

Dienst

Siehe *Service*.

Distributed Computing

Im Distributed Computing bestehen Anwendungen aus mehreren miteinander vernetzten Computern, die sich die Bearbeitung einer gemeinsam zugewiesenen Berechnung bzw. Aufgabe untereinander teilen [50]. Für den Benutzer erscheint diese Sammlung voneinander unabhängig funktionierender Computer dabei als ein einzelnes homogenes System.

Die zum Distributed Computing gehörenden Techniken und Verfahren der Kommunikation und Kooperation von Funktions- und Realisierungskomponenten bilden die technische Basis für die Teile des Mobile- wie auch das Ubiquitous Computing, die auf miteinander vernetzten Komponenten basieren. Besonders die im Bereich der Weitverkehrsnetze und dem darauf abgestützten Wide-Area-(Distributed-)Computing behandelte Kooperation heterogener Komponenten ist für mobile und ubiquitäre Systeme als Grundlage für die ad hoc Kooperation von hoher Bedeutung.

Informationstechnologie

Allgemeinbezeichnung für die Technik der Informations- und Datenverarbeitung. Darin enthalten sind beispielsweise Verfahren zur Verarbeitung von Informationen und Daten aber auch die Telekommunikation.

Intelligentes Haus

Unter dieser Bezeichnung versteht man allgemein die Integration von softwarebasierter Regelungstechnik aus der Gebäudeautomation mit einem verteilten Informationssystem. Die verwendeten Regelungsmechanismen und Interaktionsmöglichkeiten zwischen System und Benutzer können dabei sehr aufwändige Formen annehmen, bis hin zur Verwendung von Methoden der

KI. Das Verhalten eines solchen Systems dadurch zeitweise von seinem Benutzer bei oberflächlicher Betrachtung als intelligent empfunden werden.

Nutzbarkeit von Anwendungen

Eine Anwendung ist einer bestimmten Situation nutzbar, wenn die notwendige Infrastruktur aus Hard und Software in dieser Situation für die Anwendung verfügbar gemacht werden kann, die Anwendung die momentanen Benutzerbedürfnisse erfüllen kann (d.h. anwendbar ist) und die dafür notwendigen Interaktionen nicht mit der Benutzersituation in Konflikt stehen, also den momentan freien Interaktionsfähigkeiten entsprechen. Letzteres bedeutet, die Anwendung ist im weitesten Sinne (d.h. bis hin zu einer vollständigen Automatisierung) bedienbar.

Kontext

Kontext ist die hinreichend genaue Charakterisierung der Situationen eines Systems anhand von für die Adaption dieses Systems relevanten und vom System wahrnehmbaren Informationen.

Kontextadaption

Kontextadaption ist eine explizite Anpassung des beobachtbaren Verhaltens oder des inneren Zustandes eines Systems an seinen Kontext.

KI

Das Gebiet der Künstlichen Intelligenz dreht sich im Wesentlichen um die Frage, wie es Computern möglich ist, vergleichbar dem Menschen innerhalb einer komplexen Welt zu agieren, also sich wie ein Mensch zu verhalten oder wie ein Mensch zu denken. Grundlegendes Ziel ist, die Nützlichkeit von Computern durch das Studium von Prinzipien, welche die Intelligenz ermöglichen, zu verbessern [47]. Einzelne Prinzipien umfassen die Gebiete der Wissensrepräsentation, Schlussfolgerung und Problemlösung.

Mobile Computing

Diese Form der Computernutzung geht vor allem einher mit der Verbreitung von Mobiltelefonen und der damit verbundenen Verfügbarkeit weiträumiger drahtloser Vernetzung. Gleichzeitig existieren Konvergenzbestrebungen, Telefonie, Multimedia und Datendienste miteinander zu verschmelzen und dadurch neue Anwendungsfelder für den Mobilfunk zu erschließen. Das engere Forschungsgebiet Mobile Computing wird normalerweise mit der Mobilität von Hardware, Daten und Software in Computeranwendungen assoziiert.

Modell

Ein Modell ist eine idealisierte, vereinfachte [...] Darstellung eines Gegenstandes, Systems oder sonstigen Weltausschnitts mit dem Ziel, daran bestimmte Eigenschaften des Vorbilds besser studieren zu können [15]

Nomadic Computing

Diese Form der Computernutzung basiert auf der Idee eines Benutzers, der sich zwischen verschiedenen Infrastrukturen bewegt und die jeweils an einem Ort vorgefundene Infrastruktur dazu benutzt, sich mit seinem Heimatsystem zu verbinden und dessen Dienste zu nutzen. Das Nomadic Computing konzentriert sich also darauf, die gleiche Anwendung an jedem Ort in gleicher Weise transparent, d.h. ohne ortsspezifische Bedienungsschritte zur Verfügung zu stellen [44].

Persistence Problem

Das generelle Problem der Vorhersage von Eigenschaften, die durch die Ausführung einer Aktion nicht berührt werden. Bezeichnung für die Erweiterung des ursprünglichen Frame-Problems des Situation Calculus auf beliebige Kalküle (die zum Beispiel auch nebenläufige Aktionen erlauben) [32].

Personal Computer

Ein Arbeitsplatzrechner oder Personal-Computer ist ein durch eine Person bedienbares und auf deren eigene Bedürfnisse hin anpassbares Computersystem.

Prozess

Prozesse sind eine gängige und intuitiv verständliche Betrachtungsweise von Vorgängen, die einen definierten Anfang und (mit Ausnahme von unendlichen Prozessen) auch ein Ende haben und ein Ergebnis produzieren.

Qualification Problem

Beschreibt das Problem der Spezifikation, welche Bedingungen in der Welt erfüllt sein müssen, damit eine Aktion ihren gewünschten Effekt hat [32]. Zum Beispiel welche Annahmen korrekt sein müssen, damit das Drehen eines Zündschlüssels auch tatsächlich den Motor eines Autos startet.

Ramification Problem

Das Ramification Problem bezieht sich auf die Schwierigkeit der Beschreibung, was sich indirekt als Folge einer bestimmten Aktion verändert. Dabei stehen besonders Effekte im Vordergrund, die sich erst mit einer gewissen Zeitverzögerung einstellen [32]. Zum Beispiel kann das Öffnen der Jalousien

dazu führen, dass sich ein bestimmter Raum später am Tag durch die direkte Sonneneinstrahlung stark erwärmt.

Robotik

Robotik beschäftigt sich mit der Realisierung möglichst flexibel einsetzbarer "intelligenter" Maschinen und ist damit auch für Software ein wichtiges Anwendungsfeld, in dem viele zentrale Gebiete der Informatik, wie beispielsweise Mustererkennung, Künstliche Intelligenz, Steuern und Regeln von Aktuatoren, Aufbau von Sensorsystemen und einiges mehr zusammenfließen und auch zum Teil interdisziplinär an Themenbereiche aus benachbarten Ingenieurwissenschaften, wie der Elektrotechnik und dem Maschinenbau anknüpfen.

Script

In der Informatik bezeichnet ein Script ein einfaches, meist direkt über den Sourcecode ausführbares Programm in einer Hilfs- oder applikationsspezifischen Sprache.

Service

Ein Service ist die Beschreibung einer Funktionalität, eines Zugriffsprotokolls und opti-onal weiterer funktionaler Abhängigkeiten. Ein Service dient zur Spezifikation und Ab-bildung von Funktionalen, Architektur- oder Implementierungsflexibilitäten innerhalb eines Komponentensystems. Eine konkrete Abbildung auf eine laufende Komponente heißt Service Instanz oder Agent.

Im Falle der Verwendung eines Service kann man zwischen vier Hauptaktivitäten unterscheiden: Die Erzeugung eines Service, das Publizieren, das Auffinden und dem Aufruf eines Service.

Diesen Aktivitäten können verschiedenen Rollen zugeordnet werden, die diese jeweils durchführen. Der Service Anbieter (Provider) erzeugt und publiziert den Service, der Nutzer sucht und benutzt einen Service und der Vermittler (Broker), der die Vorgänge des Publizierens und Auffindens für die anderen beiden Rollen durchführt.

Jede dieser Rollen hat verschiedene, jeweils auf die Tätigkeit spezialisierte Sichtweisen auf einen Service. Für das Auffinden werden besonders Informationen über die Art der Funktionalität benötigt, welche gesucht werden soll. Diese Sicht heißt Prozess-Sicht. Für die Nutzung eines Service wird eine detaillierte Beschreibung der technischen Schnittstelle benötigt. Diese Sicht heißt Technik-Sicht. Für die Erzeugung eines Service muss zusätzlich be-

kannt, sein, von welchen anderen Funktionalitäten der gewünschte Service abhängt. Diese Sicht heißt Entwicklungs-Sicht.

Situation

Eine Situation ist die augenblickliche Lage, die Verhältnisse, die Umstände oder der allgemeine (objektive) Zustand, in dem sich jemand (oder etwas) befindet.

Smart Space

Ein Ort an dem über ein oder mehrere Objekte der realen Welt mit Softwareprozessen interagiert werden kann. Die Objekte müssen dazu nicht unbedingt als ein Computerterminal erkennbar und können in ihrer Funktion jeweils stark eingeschränkt sein. In der Regel bestehen Smart Spaces aus einer größeren Anzahl solcher mit Computern ausgestatteter Elemente [93].

Spontane Hülle

Die Spontane Hülle definiert Aufgaben eines menschlichen Benutzers, die eine begrenzte Anpassung des Systems an die jeweilige Einsatzsituation enthalten.

System

Ein System ist ein abstrakter und aus einer bestimmten Perspektive explizit von seiner Umgebung abgegrenzter Gegenstand [15]. Systeme sind aus Teilen (Systemkomponenten oder Subsystemen) zusammengesetzt, die untereinander in verschiedenen Beziehungen stehen können. Nicht weiter zerlegbare Systemteile werden als Systemelemente bezeichnet [14]. Bezogen auf die physikalische Realität ist ein System ein "Teil der objektiven Realität, der von seiner Umwelt abgegrenzt betrachtet wird, indem nur wesentliche Wechselwirkungen mit ihr betrachtet werden, von unwesentlichen aber abgesehen wird. Alle konkreten Systeme mit gleicher Verhaltensweise definieren ein abstraktes oder kybernetisches System. Die Abgrenzung des Systems ist allerdings stets ein Eingriff in die dialektische Einheit des Universums und führt zwangsläufig zu einem Verlust an Information. Die nicht als wesentlich ausgewählten Beziehungen zur Umwelt treten als Zufallserscheinungen auf." [96]

Temporal Projection Problem

Die Erweiterung des Persistence Problems umd das Problem zu bestimmen, wie sich Dinge über die Zeit hinweg verändern können. [32]

Ubiquitous Computing

Ubiquitous Computing ist die direkte oder indirekte Nutzung computerbasierter Anwendungssysteme in möglichst vielen Situationen (z.B. Orte, Zeiten, Tätigkeiten) eines Benutzers.

Wearable Computing

Neue Form der Mensch-Computer Schnittstelle, die auf Geräten basiert, die ständig in Benutzungsbereitschaft und verfügbar sind. Oft wird neben der Portabilität und dem kontinuierlichen Betrieb noch das Merkmal einer freihändigen Bedienung gefordert [42]. Als Forschungsgebiet beschäftigt sich das Wearable Computing mit ähnlichen Fragestellungen der Mobilität wie das Mobile Computing, jedoch tritt die technische Interaktion mit wechselnden heterogenen Infrastrukturen in den Hintergrund. Meist trägt der Benutzer eine aufeinander abgestimmte homogene Ausrüstung, die als Body Area Network kommuniziert und die meisten Funktionen aus der Sicht des Benutzers autark erbringt und höchstens über ein drahtloses Netz auf nicht mehr transportable oder aktualisierte Daten zugreift. Dafür rückt der Aspekt der Interaktion mit dem Benutzer stärker in den Vordergrund, da das System permanent am oder sogar im Körper getragen wird.

A5 Zeichenerklärung

$\lvert x \rvert$	Kardinalität einer Menge x
$\lVert t \rVert$	Menge aller Nachrichten vom Typ t
$\mathbb{P}$	Potenzmengenoperator
M^*	Menge der endlichen Sequenzen über **M**
M^∞	Menge aller Nachrichtenströme über **M**
$M^\surd$	Menge der gezeiteten Nachrichtenströme
$x\lvert_n$, für $n \in \mathbb{N}$	Sequenz der ersten i Sequenzen des Stroms $x \in M^\infty$
CH	Menge der getypten Kanäle
M	Menge der Nachrichten
$\mathbf{T_M}$	Menge der Nachrichtentypen
$\mathbf{T_{CH}}$	Menge der Kanaltypen
$\overline{C}$:**CH** $\to M^\infty$	Bündel von Strömen
CH	Kanalbelegung
$\otimes$	Kompositionsoperator für Komponenten
$\sqsupseteq$	Präfix
x $\triangleleft$ y	Filtert Nachrichten aus der Menge x aus dem Strom y
$C(K)$, *für* $K \subseteq$ **CH**	Menge der Komponenten über K
$A(K)$, *für* $K \subseteq$ **CH**	Menge der Agenten über K
$x\lvert_{C'}$ $\overline{C}'$	Einschränkung von $x \in \overline{C}$ auf $C' \subseteq C$
$a\&s$	Konkatenation

Index

A

B

C

D

E

F

G

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Abbildungsverzeichnis

Definitionsverzeichnis

Quelltextverzeichnis

Literaturverzeichnis

[1] K. Warwick: Cyborg 1.0. *Wired Magazine* Issue 8.02, Feb. 2000. http://www.wired.com/wired/archive/8.02/warwick.html

[2] Dictionary.com. Online Webservice. http://www.dictionary.com

[3] *Longman Dictionary of Contemporary English*. Langenscheidt, 2. Auflage 1987.

[4] G. Wahrig: *Wahrig Deutsches Wörterbuch*. Bertelsmann LexikonVerlag, Gütersloh/München 2000.

[5] Brockhaus - Die Enzyklopädie. Online Webservice. http://www.brockhaus.de

[6] Microsoft Launches Smart Personal Object Technology Initiative. PressPass Information for Journalists, Redmond, Wash., Microsoft, Nov. 2002 http://www.microsoft.com/presspass/features/2002/nov02/11-17spot.asp

[7] T. Murakami, A. Fujinuma: Ubiquitous Computing: Towards a New Paradigm. NRI Papers No.2, Nomura Research Institute, April 1, 2000. http://www.nri.co.jp/english/opinion/papers/2000/pdf/np200002.pdf

[8] M. Weiser. Ubiquitous Computing Homepage, Xerox Parc. http://nano.xerox.com/hypertext/weiser/UbiHome.html

[9] M. Esler, J. Hightower, T. Anderson, G. Borriello: Next Century Challenges: Data-Centric Networking for Invisible Computing. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 256-262, ACM Press, Seattle, WA USA, August 1999. http://www.acm.org/pubs/articles/proceedings/comm/313451/p256-esler/p256-esler.pdf

[10] G. Dodig-Crnkovic: Scientific Methods in Computer Science. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*. Skövde, Sweden. April 2002. http://www.mrtc.mdh.se/publications/0446.pdf

[11] H Balzert: *Lehrbuch der Objektmodellierung (Analyse und Entwurf)*. Spektrum Akademischer Verlag, 1999.

[12] C. Salzmann: *Modellbasierter Entwurf spontaner Komponentensysteme*. Dissertation, Technische Universität München, 2002.

[13] Arbeitskreis "Terminologie der Softwaretechnik" der Fachgruppe 2.1.1: Tätigkeiten und Ergebnisse der Analyse. Online Publikation. http://www.tfh-berlin.de/~giak/arbeitskreise/softwaretechnik/themenbereiche/analyse.html

[14] Arbeitskreis "Terminologie der Softwaretechnik" der Fachgruppe 2.1.1: Grundbegriffe. Online Publikation. http://www.tfh-berlin.de/~giak/arbeitskreise/softwaretechnik/themenbereiche/grundbgr.html

[15] W. Hesse, G. Barkow, H. v. Braun, H.-B. Kittlaus, G. Scheschonk: Terminologie in der Softwaretechnik - Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen. Teil 1: Begriffssystematik und Grundbegriffe. In: *Informatik-Spektrum* 17, pp. 39-47, Springer-Verlag, 1994.

[16] W. Hesse, G. Barkow, H. v. Braun, H.-B. Kittlaus, G. Scheschonk: Terminologie in der Softwaretechnik - Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen. Teil 2: Tätigkeits- und ergebnisbezogene Elemente. In: *Informatik-Spektrum* 17, pp. 96-105, Springer-Verlag, 1994.

[17] H. v. Braun, W. Hesse, H.-B. Kittlaus, G. Scheschonk: Ist die Welt objektorientiert? - Von der natürlich-sprachlichen Weltsicht zum OO-Modell. In: Tagungsband *Natürlichsprachlicher Entwurf von Informationssystemen - Grundlagen, Methoden, Werkzeuge, Anwendungen*. Workshop, Ev. Akademie Tutzing, 28.-30. Mai 1996.

[18] G. Barkow, W. Hesse, H.-B. Kittlaus, A.L. Luft, G. Scheschonk, A. v. Stülpnagel: Begriffliche Grundlagen für die frühen Phasen der Software Entwicklung. In: *Information Management* 4/89, pp. 54-60, 1989.

[19] D. Brade, I. Cegla: Conceptual Modelling Meets Formalization. In *Proceedings of the 03 Spring Simulation Interoperability Workshop*, organized by the Simulation Interoperability Standardization Organization, Orlando, 2003.

[20] J. Raasch: *Systementwicklung mit Strukturierten Methoden. Ein Leitfaden für Praxis und Studium*. Hanser, 3. Auflage, München, Wien, 1993.

[21] A. K. Dey, G. D. Abowd: CybreMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing* (HUC2K), pp. 172-186, Bristol, UK, September 25-27, 2000.
http://www.cc.gatech.edu/fce/contexttoolkit/pubs/HUC2000.pdf

[22] N. Sawhney, S. Wheeler, C. Schmandt: Aware Community Portals: Shared Information Appliances for Transitional Spaces. In *Journal of Personal and Ubiquitous Computing*, Vol. 5 Issue 1, pp. 66-70. Springer-Verlag, February 2001.
http://web.media.mit.edu/~nitin/portals/chi00/portals_chi_wkshp.pdf

[23] C. 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

[24] M. Jedamzik: Intelligentes Haus. Ein ergonomisches Dialogsystem zur Steuerung von technischen Systemen in Wohnbereichen mittels Gestenerkennung. Online Publikation.
http://ls7-www.cs.uni-dortmund.de/research/gesture/argus/intelligent-home

[25] Innovationszentrum Intelligentes Haus Duisburg. Homepage.
http://www.inhaus-duisburg.de

[26] A. K. Dey: *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.
http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf

[27] B. Rhodes: The Wearable Remembrance Agent: A system for augmented memory. In *Personal Technologies Journal Special Issue on Wearable Computing*, 1:218-224, Personal Technologies, 1997.
http://xenia.media.mit.edu/~rhodes/Papers/wear-ra-personaltech/index.html

[28] H. Lieberman, T. Selker: Out of context: Computer Systems that adapt to, and learn from, context. In *IBM Systems Journal*, Vol. 39, Nos 3&4, p. 617-632, 2000.
http://www.research.ibm.com/journal/sj/393/part1/lieberman.pdf

[29] B. Rhodes: *Just-In-Time Information Retrieval*. PhD thesis, MIT Media Laboratory, Cambridge, MA, June 2000.
http://xenia.media.mit.edu/~rhodes/Papers/rhodes-phd-JITIR.pdf

[30] W. N. Schilit: *System architecture for context-aware mobile computing*. PhD thesis, Columbia University, 1995.
http://seattleweb.intel-research.net/people/schilit/schilit-thesis.pdf

[31] S. Shafer: Ten Dimensions in Ubiquitous Computing. Keynote presentation at Conference on Managing Interactions in Smart Environments, December 1999.

[32] L. Morgenstern. The problem with solutions to the frame problem. Technical report, IBM T.J. Watson Research Center, 1995.
http://citeseer.ist.psu.edu/morgenstern95problem.html

[33] Windows Powered Smart Displays: Windows XP Anywhere in the Home. Online Publikation. Microsoft.
http://www.microsoft.com/windowsxp/smartdisplay/default.asp

[34] M. Weiser: Ubiquitous Computing. In *IEEE Computer Hot Topics*, October 1993.
http://www.ubiq.com/hypertext/weiser/UbiCompHotTopics.html

[35] M. Weiser, J.S. Brown. Designing Calm Technology. In PowerGrid Journal, v 1.01, http://powergrid.electriciti.com/1.01, July 1996.
http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm

[36] TeleAid - The Emergency-Call System that Automatically Calls for Help. Online Publikation.
http://www.mercedes-benz.com/e/innovation/rd/teleaid.htm

[37] A. Moudgal: FAQ. In Ubiquitous Computing News. Online Publikation, 1999
http://ubicomp.editthispage.com/faq

[38] D. Kara: Pervasive Computing Era - (Industry Trend or Event). SoftwareMag.com Issue: April, 2000.
http://www.findarticles.com/cf_dls/m0SMG/2_20/62194838/p1/article.jhtml

[39] 802.11Planet.com. 802.11 Glossary – pervasive computing. Online Publikation, 2002
http://80211-planet.webopedia.com/TERM/P/pervasive_computing.html

[40] A.C. Huang, B. Ling, S. Ponnekanti, and A. Fox: Pervasive Computing: What Is It Good For? In: *Workshop on Mobile Data Management* (MobiDE) in conjunction with ACM MobiCom, 1999

[41] D. Lim: Ubiquitous mobile computing: UMC's model and success. In: Educational Technology & Society 2(4), 1999
http://ifets.ieee.org/periodical/vol_4_99/dan_lim.html

[42] S. Mann: WEARABLE COMPUTING as means for PERSONAL EMPOWERMENT. In: *International Conference on Wearable Computing* ICWC-98, Fairfax VA, May 1998.
http://wearcam.org/icwckeynote.html

[43] What is a Wearable? Online Publikation, MIT MediaLab, 2003
http://www.media.mit.edu/wearables/

[44] R. H. Katz: Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, Volume 1, pages 6—17, 1994

[45] W Wagner: Einführung in die Methoden der empirischen Sozialforschung. 2. Sitzung: Theorie, Hypothese und Operationalisierung. Online Publikation
http://www.fh-erfurt.de/so/wagner/1empsitz02theorie.html

[46] J. Pascoe: Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers* (ISWC'98), pp. 92-99, Pittsburgh, PA, IEEE, October 19-20 1998.
http://www.cs.ukc.ac.uk/pubs/1998/676/content.zip

[47] P.H. Winston: *Artificial intelligence*. Addison-Wesley Verlag Reading, 1993.

[48] A. Pentland: Wearable Intelligence, *Scientific American*, Vol. 276, No. 1es1, Nov. 1998.

[49] M. Fahrmair, C. Salzmann, M. Schoenmakers: Verfahren zur Vorauswahl mobiler Dienste. Patent DE0010024368A1 [DE], Deutsches Patentamt, 2000

[50] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY: 1990.

[51] The UPnP Consortium. Universal plug and play specification. Online Publikation, 2000
http://www.upnp.org

[52] The Home Automation Forum. List of recommended software. Online Publikation.
http://www.homeautomationforum.com/links/Software/

[53] Control your Home from anywhere! Homepage. HomeSeer Technologies LLC.
http://www.homeseer.com/

[54] MisterHouse - It Knows Kung-Fu. Homepage.
http://misterhouse.net:81/

[55] X10 Theory and Protocolls. Online Publikation.
http://www.geocities.com/ido_bartana/toc.htm

[56] MIT Project Oxygen. Pervasive, Human-Centered Computing. Homepage.
http://oxygen.lcs.mit.edu/index.html

[57] D. Quan, D. Huynh, D.R. Karger. Haystack: A Plattform for Authoring End User Semantic Web Applications. In: *IEEE International Symposium on Wearable Computers* (ISWC), 2003
http://haystack.lcs.mit.edu/papers/iswc2003-haystack.pdf

[58] Nat Friedman. The Dashboard. Online Publikation, July 2003.
http://www.nat.org/dashboard/ols2003/ols2003-dashboard.ppt

[59] N. Houssos, A. Alonistioti, L. Merkakos, E. Mohyeldin, M. Dillinger, M. Fahrmair, M. Schoenmakers: Advanced Adaptability and Profile Management Framework for the Support of Flexible Mobile Service Provison. *IEEE Wireless Communications Mag*, August 2003

[60] M. Dertouzos: *The Unfinished Revolution: Human-Centered Computers and What They Can Do for Us*, NY, HarperCollins Publishers, 2001

[61] Salber, Daniel, A. K. Dey, G. D. Abowd: The Context Toolkit: Aiding the development of context-enabled applications. In *Proceedings of the 1999 ACM Conference on Human Factors in Computer Systems* (CHI '99), pp. 434-441, Pittsburgh, PA, ACM. May 15-20, 1999.
http://www.cc.gatech.edu/fce/contexttoolkit/pubs/chi99.pdf

[62] M. 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.

[63] M. Broy and K. Stolen: *Specification and Development of Interactive Systems - Focus on Streams, Interfaces and Refinement*. Monographs in Computer Science. Springer Verlag 2000.

[64] L. Hilty et. al.: Das Vorsorgeprinzip in der Informationsgesellschaft - Auswirkungen des Pervasive Computing auf Gesundheit und Umwelt. TA-SWISS Zentrum für Technologiefolgenabschätzung, 2003. http://www.ta-swiss.ch/www-remain/reports_archive/publications/2003/030904_PvC_Bericht.pdf

[65] H. Prantl: Lauschangriff. Wenn der Wanzenmann kommt. *Süddeutsche Zeitung* 9.7.2003. http://www.sueddeutsche.de/deutschland/artikel/224/14210/

[66] D. 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. http://www2.psych.cornell.edu/andrews/fws01/dennett_frame.html

[67] Pamela Zave: FAQ Sheet on Feature Interaction. Online Publikation, AT&T, 1999. http://www.research.att.com/~pamela/faq.html

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

[69] Nano. Wohnen im "intelligenten Haus". Wie sinnvoll ist die Vernetzung von Haushaltsgeräten? Fernsehbericht, 3SAT 03.03.2001. http://www.3sat.de/3sat.php?http://www.3sat.de/nano/astuecke/17543/

[70] W. Stieler: Fürsorgliche Belagerung, *C't Magazin für Computertechnik* 5/2004, Heise Verlag, 2004.

[71] M. Breitling: *Formale Fehlermodellierung für verteilte reaktive Systeme*. Dissertation, Technische Universität München, Fakultät für Informatik, 2001.

[72] R. Pfeifer,P. Rademakers: 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 1991.

[73] S. Greenberg: Context as dynamic construct. *Human-Computer Interaction*, 16(2-3), 2001.

[74] J. McCarthy, P. J. Hayes: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence* 4, ed. D.Michie and B.Meltzer, Edinburgh: Edinburgh University Press, pp. 463-502, 1969.

[75] J. v. Brakel: The Complete Description of the Frame Problem. PSYCOLOQUY 3(60) frame-problem.2, 1992

[76] J. H. Fetzer: Van Brakel's Position Appears to be Incoherent. PSYCOLOQUY 4(14) frame-problem.4, 1993

[77] S. Harnad: The Symbol Grounding Problem. *Physica* D 42: 335-346. 1990.

[78] S. Harnad: Problems, Problems: the Frame Problem as a Symptom of the Symbol Grounding Problem (11), Psycoloquy, 1993 http://psycprints.ecs.soton.ac.uk/archive/00000328/

[79] A. Kofod-Petersen, A. Aamodt: Case-Based Situation Assessment in a Mobile Context-Aware System. In: *Artificial Intelligence in Mobile Systems* (AIMS), 2003. http://w5.cs.uni-sb.de/~krueger/aims2003/camera-ready/kofod-petersen-6.pdf

[80] L. A. Suchman: *Plans and Situated Actions - The Problem of Human-Machine Communication*. Cambridge University Press 1987.

[81] W. J. Clancey: *Situated cognition*. Cambridge University Press. 1997

[82] R. Rosnow: Whatever happened to the "Law of Primacy.". In: *Journal of Communication*, 16, 10-31, 1966.

[83] The Frame Problem. In: Psycoloquy (ISSN 1055-0143), 1992-1993. http://psycprints.ecs.soton.ac.uk/view/topics/frame-problem.html

[84] K.M. Colby, et al: Turing-like undistinguishability tests for the validation of a computer simulation of paranoid processes. *Artificial Intelligence*, 3, 47-51, 1973.

[85] M. C. Mozer: Lessons from an adaptive house. In D. Cook & R. Das (Eds.), *Smart environments: Technologies, protocols, and applications*. J. Wiley & Sons. 2004. ftp://ftp.cs.colorado.edu/users/mozer/papers/smart_environments.pdf

[86] Futurelife. Homepage. http://www.futurelife.ch/

[87] R. Johnson, B. Foote: Designing Reusable Classes. *Journal of Object-Oriented Programming*. SIGS, 1, 5 (June/July. 1988), 22-35

[88] G. Krasner, S. Pope.: A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80. Issue of the *Journal of Object-Oriented Programming (JOOP)*, August/September 1988

[89] 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.

[90] TRUST Project. Homepage. http://www4.in.tum.de/~scout/trust_webpage_src/trust_frameset.html

[91] Scout Project. Homepage. http://www.ist-scout.org

[92] E2R Project. Homepage. http://www.e2r.motlabs.com

[93] J. Heidemann, R. Govindan, D. Estrin: Configuration Challenges for Smart Spaces. Technical Report 98-677, University of Southern California, July, 1998. originally appeared in the DARPA/NIST Smart Spaces Workshop. http://www.isi.edu/~johnh/PAPERS/Heidemann98e.html

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

[95] E. Mohyeldin, M. Dillinger, M. Fahrmair, P. Dornbusch, 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, .

[96] *Fachlexikon Physik*, Harri Deutsch Verlag, Leipzig 1982.

[97] I. Sutherland: A head-mounted three dimensional display. In *Proc. Fall Joint Computer Conference*, pages 757-764, 1968.

[98] R. T. Azuma: A survey of augmented reality. In *Computer Graphics* (SIG-GRAPH '95 Proceedings, Course Notes #9: Developing Advanced Virtual Reality Applications) (Aug. 1995), pp. 1--38.

[99] Adaptation Definition. The Everyday Spelling Homepage. Online Publikation, Scott Foresman-Addison Wesley. http://www.everydayspelling.com/workout/wordsnews/adaptation_definition.html

[100] R. Milner, J. Parrow, and D. Walker: *A calculus for mobile processes. Information and Computation*, 100:1–77, 1992.

[101] L. Cardelli, A. D. Gordon: Mobile ambients. Technical report, Microsoft Research LTD., 1998.

[102] D. McDermott: A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6, 101-155, 1982.

[103] M. Fahrmair, E. Mohyeldin, C. Salzmann: Communication Profiles for Reconfigurable Systems. In: Dillinger, Madani, Alonistioti: *Software Defined Radio: Architecture, Systems and Functions*. John Wiley & Sons, 2003.

[104] N. Diernhofer, M. Fahrmair, U. Eschbach: Intelligente Portale für mobile Dienste. *Mobile Computer & Communications* 10/2002.