Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011)

Ableitung von Anforderungen zum Adaptionsverhalten in ubiquitären adaptiven Anwendungen

Christoph Evers, Axel Hoffmann, Daniel Saur,
Kurt Geihs und Jan Marco Leimeister

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Ableitung von Anforderungen zum Adaptionsverhalten in ubiquitären adaptiven Anwendungen

Christoph Evers[1], Axel Hoffmann[2], Daniel Saur[3], Kurt Geihs[4] und Jan Marco Leimeister[5]

[1]evers@vs.uni-kassel.de, Fachgebiet Verteilte Systeme
[2]axel.hoffmann@uni-kassel.de, Fachgebiet Wirtschaftsinformatik
[3]saur@vs.uni-kassel.de, Fachgebiet Verteilte Systeme
[4]geihs@uni-kassel.de, Fachgebiet Verteilte Systeme
[5]leimeister@uni-kassel.de, Fachgebiet Wirtschaftsinformatik

Forschungszentrum für Informationstechnik-Gestaltung
http://www.uni-kassel.de/iteg
Universität Kassel

Zusammenfassung: Das Adaptionsverhalten in selbst-adaptiven Anwendungen in der Softwareentwicklung wird, wie auch andere Funktionalität, durch Anforderungen bestimmt. Wirkt sich das Adaptionsverhalten auf die Anwendungsarchitektur aus, so werden die zugehörigen Anforderungen bereits am Anfang der Umsetzung benötigt. Diese Arbeit beantwortet die Frage, welche Anforderungen erfasst werden müssen und wie diese systematisch hergeleitet werden können. Das Vorgehen lehnt sich dabei an die Entwicklungsmethodik und das Rahmenwerk des MUSIC-Projektes an und wird anschließend an einem Fallbeispiel demonstriert. Dieses lässt sich ebenfalls für andere Entwicklungsprojekte und -ansätze übernehmen, deren Ziel es ist, selbst-adaptive Anwendungen zu erstellen. Mit dem Vorgehen ist eine effektive Ausarbeitung des Adaptionsverhalten anhand von Anforderungen bereits zu Beginn der Umsetzung möglich.

Stichwörter: Selbst-adaptive Systeme, Anforderungsanalyse, Model-Driven-Development, MUSIC

1 Einleitung

Mobile und Ubiquitous Computing erhöhen die Notwendigkeit nach Anwendungen, die in der Lage sind, sich dynamisch an die Umgebung, Ressourcen und Nutzerpräferenzen anzupassen. Beispiele sind wechselnde Netzwerkverbindungen, schwankende Akkustände, die Einbindung neuer Geräte und Dienste, sowie wechselnde Nutzer mit unterschiedlichen Präferenzen. Solche Adaptionen werden durch die automatische Einbindung von unterschiedlichen Varianten einer Anwendung erreicht, welche die gleiche Basisfunktionalität mit wechselnder Dienstqualität anbieten oder auch gänzlich alternative Konfigurationen darstellen.

Für Anwendungen, die auf eine komponentenbasierte Architektur aufbauen und externe Dienste integrieren, können die Anwendungsvarianten mit Hilfe eines Variablititätsmodell erstellt werden. Eine Anwendung wird aus Komponenten oder Diensten zusammengesetzt, wobei jeder Be-

standteil ein Vielzahl von Varianten haben kann. Daher ist die Gesamtzahl der Anwendungsvarianten das Produkt der Varianten der einzelnen Bestandteile. In einer Ubiquitous Computing Umgebung können Komponenten und Dienstleistungen erscheinen und während der Laufzeit wieder verschwinden. Diese Tatsache spricht gegen eine statische Architektur der Anwendung, was bedeutet, dass mindestens eine Variante jeder Komponente vorhanden sein muss.

Um die Zusammensetzung der Anpassung zu realisieren, muss eine Middleware in der Lage sein verschiedene Einsatzvarianten zu bewerten und im Hinblick auf die benötigten Eigenschaften unterschiedliche Kompositionen zu realisieren. Zu diesem Zweck werden die Anwendungen auf einem komponentenbasierten Rahmenwerk aufgebaut, das eine automatische Generierung unterschiedlicher Versionen der Anwendung leisten kann.

Die Anforderungsanalyse für selbst-adaptive Software unterscheidet sich signifikant von der klassischer Anforderungsanalyse, denn die Variabilität einer Anwendung spiegelt sich meist schon in der Architektur wieder [HHL10]. Diese ist jedoch getrennt von der eigentlichen Funktionalität der Anwendung zu betrachten. Aus diesem Grund ist es notwendig eine spezielle Vorgehensweise zur Extrahierung der Anforderungen an das Adaptionsverhalten aus den Gesamtanforderungen zu verfolgen, um Variabilität bereits bei der Architekturgestaltung zu berücksichtigen.

Nach einer Einführung in selbst-adaptive Software und in die Anforderungsanalyse wird in Kapitel 4 ein Rahmenwerk zur Entwicklung selbst-adaptiver Anwendungen vorgestellt. Anhand dessen wird dann in Kapitel 5 gezeigt, wie Anforderungen für das Design der Adaptivität aus den Anforderungen der Anwendung extrahiert werden können. Kapitel 6 zeigt dann das Vorgehen an einem konkreten Beispiel.

2 Selbst-adaptive Software

Selbst-Adaption beschreibt die Fähigkeit eines Software- oder Hardwaresystems sich selbstständig zur Laufzeit an sich verändernde Umgebungen und Situationen anzupassen. Diese Notwendigkeit ist insbesondere dann gegeben, wenn sich Menschen und Informationstechnologie in einem hoch dynamischen und mobilen Kontext bewegen. Um diese Dynamik abbilden zu können, ist die Komplexität von Anwendungen in diesen Anwendungsfeldern meist höher als in statischen Umgebungen. Selbst-adaptive Systeme versuchen diese Komplexität zu verringern, indem sie die richtigen Informationen und die richtige Konfiguration in der jeweiligen Situation bereitstellen. Sie sind in der Lage Kontextänderungen zu erkennen und auf diese in einer geeigneter Form zu reagieren, ohne dass der Benutzer unnötig eingreifen muss bzw. irritiert wird. Darüberhinaus gib es Situationen, die bei der Planung und Entwicklung einer Anwendung nicht vorgesehen werden konnten. Eine adaptive Anwendung ist in der Lage auf derartige Situation angemessen zu reagieren, indem verfügbare Dienste eingebunden werden oder Datenformate transformiert werden können. Insbesondere bei der Nutzung von mobilen Anwendungen ist es wünschenswert eine Kontinuität der Anwendungsausführung, auch bei sich stetig änderndem Kontext, zu haben. Adaption an den jeweiligen Kontext versucht genau dieses Problem zu adressieren. Die Komplexität moderner Software erschwert außerdem die Ausführung umfangreicher Anwendungen auf ressourcenbeschränkten mobilen Geräten. Eine komponentenbasierte selbstadaptive Anwendung versucht die vorhandenen Betriebsmittel wie Speicher, Bandbreite oder

Akkuleistung möglichst optimal auszunutzen, indem zur Laufzeit immer nur die Komponenten geladen werden, die auch nur tatsächlich in dieser Situation bzw. in diesem Ausführungskontext benötigt werden.

Ein weitverbreiteter Ansatz zur Entwicklung und Bereitstellung von selbst-adaptiven Anwendungen sind komponentenbasierte Middlewares [Gei09] [Rou09]. Diese meist auch reflektiven Middlewares bilden ein so genanntes geschlossenes System, in dem die Selbst-Adaption ohne jegliche Benutzerinteraktion zur Laufzeit durchgeführt wird.

Dieser Vorgang umfasst drei notwendige Schritte, die im folgenden kurz erläutert werden.

1. Überwachung der Umgebung auf Kontextänderungen.
   Eine adaptive Anwendung muss die dynamische Umgebung überwachen und gegebenfalls auf Benutzereingaben reagieren können. Die Umgebungsinformationen werden hauptsächlich von externen (Hardware-)Sensoren bereitgestellt, dessen Daten durch die Anwendung verarbeitet und analysiert werden müssen. Die hierdurch gewonnenen Informationen werden üblicherweise als *Kontext* oder *Kontextinformationen* bezeichnet.

2. Entscheidung über notwendige Konsequenzen.
   Um ein angestrebtes Ziel im aktuellen Kontext zu erreichen, muss die Anwendung bestimmen, welche Aktion als nächstes ausgeführt werden muss (auch wenn eventuell keine Änderung der Anwendung nötig ist). Dieser Entscheidungsprozess basiert entweder auf vorher festgelegten Regeln, allgemeinen Zielvorgaben oder normierten Nutzenfunktionen [KD07].

3. Umsetzung der Adaptionsentscheidung, d.h. Re-Konfiguration des Systems.
   Sobald die Anwendung feststellt, dass eine Veränderung notwendig ist, muss eine passende Aktion ausgeführt werden. Diese variiert von einfachen Anpassungen bestimmter Parameter bis hin zur vollständigen Re-Konfiguration der Software-Architektur.

3 Anforderungsanalyse

Unter Anforderungsanalyse werden die Aktivitäten verstanden, mit denen Anforderungen erhoben und in geeigneter Form für die spätere Bearbeitung und Implementierung dokumentiert werden [NE00]. Der Erfolg eines Produktes oder von Software basiert auf einer erfolgreichen Umsetzung der Kundenanforderungen. Die Anforderungsanalyse ist ein wichtiger und zugleich kritischer Bestandteil des Entwicklungsprozesses. Mehrere Studien belegen, dass die Fehler aus der Anforderungsverwaltung den gesamten Entwicklungsprozess in hohem Maß beeinflussen [FD96] [HBR02]. Derartige Fehler verursachen zusätzliche Kosten und Aufwand, wenn sie in den späteren Phasen der Entwicklung gefunden und korrigiert werden [Poh08]. Damit hat die Anforderungsanalyse einen entscheidenden Einfluss auf den Erfolg von Entwicklungsprojekten.

Berry et al. schlagen für diese Aufgabe einen vierstufigen Ansatz vor [BCZ05]. Level 1 vereint dabei die typischen Aktivitäten der Anforderungsanalyse [Poh08]. Auf Level 2 wird festgelegt, wie sich die Anwendung verhält, wenn die Umweltfaktoren eine Adaption nötig machen. Wie die Anforderung eine Änderung der Umweltbedingungen bemerkt, wird auf Level 3 festgelegt. Level 4 bestimmt das dafür nötige Adaptionsverfahren.

Bezüglich der Anforderungsanalyse weisen Cheng et al. darauf hin [Che09], dass bestehende Ansätze die Entwicklung selbst-adaptiver Software nur unzureichend unterstützen, da die für die Adaption benötigten Informationen meistens fehlen. So lässt sich nur schwer erkennen, welche Änderungen der Umwelt das System als Anlass für eine Adaption nutzen soll und was es in Situationen macht, in denen keine optimalen Bedingungen vorliegen. Demnach benötigt das System von der Umwelt unabhängige Ziele, welche die Anwendung immer bereitstellt, und Ziele, die je nach Bedingungen durch die Adaption ermöglicht werden. Dazu ist auch wichtig zu wissen, welche Einschränkungen für die Adaption vorliegen. Somit muss während der Entwicklung unterschieden werden können, welche Umweltfaktoren für die Adaption relevant sind und welche Anforderungen während der Laufzeit geändert werden können. Deswegen muss die Anforderungsanalyse mit der gegebenen Unsicherheit umgehen können, indem eingeplant wird, dass sich auch die Anforderungen an das System zur Laufzeit ändern können [Che09]. Aus diesem Grund ist es viel wichtiger, die Möglichkeiten zu erkennen, in denen Adaption in der Anwendung möglich ist, als die einzelnen Varianten der Anwendung im Voraus zu kennen.

4 Rahmenwerk für selbst-adaptive Anwendungen

Die hier vorgestellte Arbeit basiert auf dem EU-Projekt MUSIC [FP6], in dessen Rahmen ein umfangreiches, flexibles Rahmenwerk und eine Middleware zur Entwicklung von selbst-adaptiven, mobilen Anwendungen entwickelt wurde [Rou09]. Darüber hinaus umfasst MUSIC eine vollständige Entwicklungsmethodik, die einen modellgetriebenen Entwicklungsansatz verfolgt und die Vorteile dieses Ansatzes herausstellt [GRWK09]. Zusammengenommen sollen beide Elemente die Entwicklung von selbst-adaptiven Anwendungen erleichtern, indem die Anwendungsentwickler sich auf die Funktionalität ihrer Anwendung konzentrieren können und komplizierte Vorgänge wie Adaptionsverhalten und Kontextmanagement nicht selber entwickeln müssen. Innerhalb der MUSIC-Entwicklungsmethodik wird eine Enwicklungsphase zur Anforderungsanalyse definiert. Diese Phase definiert bereits ein grundlegendes Konzept, wie Anforderungen an adaptive Anwendungen gewonnen werden können. Jedoch ist das dort beschriebene Vorgehen auf sehr kleine Entwicklerteams beschränkt, die meist auch noch selbst die Idee zur Entwicklung einer selbst-adaptiven mobilen Anwendung haben. Dies liegt hauptsächlich darin begründet, dass die anvisierten mobilen Anwendungen nur eine vergleichsweise geringe Komplexität haben [Hal09]. Mit dem verstärkten Aufkommen leistungsfähiger Smartphones und sonstigen Endgeräten, wie beispielsweise Tablet-PCs, wird sich dies jedoch zunehmend ändern. Auch neuere Paradigmen wie das des Ubiquitous Computing stellen neue Herausforderungen an die Gewinnung von Anforderungen. Beispielsweise entstehen im Kontext der allgegenwärtigen Vernetzung ganz neue rechtliche Anforderungen an eine Anwendung, um Datenschutz und Privatsphäre weiterhin zu gewährleisten. Im weiteren Verlauf wird zunächst erläutert, wie die Anforderungsanalyse innerhalb der MUSIC-Entwicklungsmethodik vorgesehen ist. Anschließend werden die aus den Anforderungen erzeugten Modelle (im Sinne der modellgetriebenen Entwicklungsmethodik) vorgestellt. Diese Modelle dienen dem hier vorgestellten Ansatz als Referenz. Auch wenn sich dieser Ansatz sehr stark an der MUSIC-Entwicklungsmethodik orientiert, so wird er auch auf andere Vorgehensweisen übertragbar sein, bei denen es darum geht, Anforderungen für selbst-adaptive Anwendungen zu gewinnen.

Der Fokus der MUSIC-Entwicklungsmethodik liegt auf der Variabilität der Anwendung, d.h. dass sowohl die Entwicklungsmethodik als auch Rahmenwerk bzw. Middleware den Entwickler, bei der Implementierung von adaptivem kontextbewusstem Verhalten in die Anwendung, unterstützt. Es gibt dabei eine strikte Trennung zwischen der Software-Architektur, die die Variabilität abbildet, und der eigentlichen Funktionalität der Anwendung. Die Entwicklung der Funktionalität wird nicht explizit durch MUSIC unterstützt, sondern wird getrennt von der Variabilität entwickelt. Es bleibt dem Entwickler überlassen, ob er auch dort einen modellgetriebenen Entwicklungsansatz verfolgen möchte. Diese Besonderheit spiegelt sich dann in der Gewinnung von Anforderungen wieder, da dort zwischen Anforderungen an die Funktionalität und Anforderungen an das Adaptionsverhalten unterschieden werden muss.

4.1 Anforderungsanalyse in MUSIC

Die Analysephase der MUSIC-Entwicklungsmethodik hat zwei grundlegende Ziele: 1) Erstellung eines Variabilitätsmodells und 2) Ableitung eines Domänenmodells [Wag10]. Die genaue Vorgehensweise ist in Abbildung 1 illustriert. Zu Beginn führt der Entwickler eine Anwendungsfall-Analyse durch, um möglichst alle notwendigen Modi der Anwendung zu identifizieren. Nachfolgend werden alle Kontextabhängigkeiten sowie benötigte Ressourcen aufgedeckt. Im Anschluss erfolgt das Identifizieren von Knotentypen sowie Diensten. Zuletzt müssen nun die eigentlichen Anwendungsmodi definiert werden.
Mit diesen Informationen erstellt der Entwickler das Variabilitätsmodell der Anwendung. Im

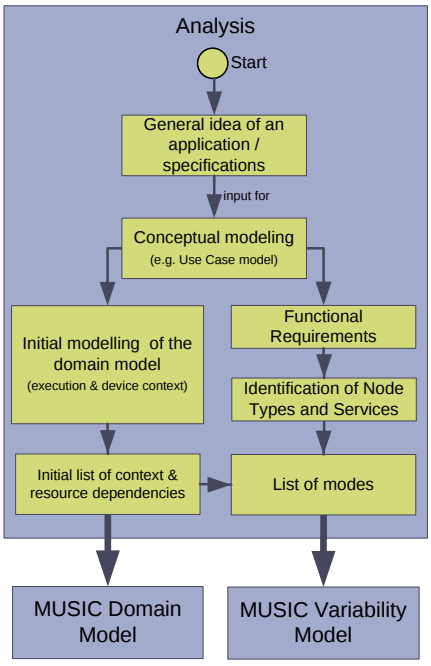

Abbildung 1: Ablauf der Analysephase in MUSIC (aus [Wag10])

MUSIC-Entwicklungsprozess spiegelt sich die Berücksichtigung aller beteiligten Rollen (Stake-

holder) in einem Softwareprojekt nicht explizit wieder. Die Anforderungen an die Applikation wurden meist durch die Entwickler selbst bestimmt. Diese Vorgehensweise stößt jedoch an ihre Grenzen, wenn 1) Software für Dritte entwickelt wird und 2) mehrere Disziplinen und Interessen gleichermaßen berücksichtigt werden müssen. Hier wird bereits deutlich, dass Anforderungen im MUSIC-Entwicklungsprozess bisher noch nicht ausreichend berücksichtigt wurden. Vielmehr geht man davon aus, dass die Anforderungen bereits in geeigneter Form vorliegen und man diese einfach in ein konzeptuelles Modell überführen kann. Ebenso wenig erkennt man hier die Einbeziehung von beteiligten und relevanten Rollen innerhalb des Projektes. Dies deckt sich auch mit den Erfahrungen, die im MUSIC-Projekt gemacht wurden: Die Pilot-Anwendungen wurden ausschließlich von wenigen Entwicklern implementiert, bei denen meist nur die eigenen Anforderungen an die Anwendung von Relevanz waren.

4.2 Modellierung von Adaptivität in MUSIC

Damit sich eine Anwendung in unterschiedlichen Ausführungskontexten (vereinfacht: Situationen) adaptieren kann, definiert MUSIC ein *Variabilitätsmodell*. Das UML-Diagramm spezifiziert eine komplette Familie von Anwendungsvarianten mit jeweils unterschiedlichen Funktionalitäten und Eigenschaften. MUSIC setzt hier auf einen modellgetriebenen Ansatz (Model-Driven Development (MDD)), bei dem das UML-Variabilitätsmodell die erste Stufe eines plattformunabhänigen Modells (PIM) darstellt [GRWK09]. Neben der konventioneller Software-

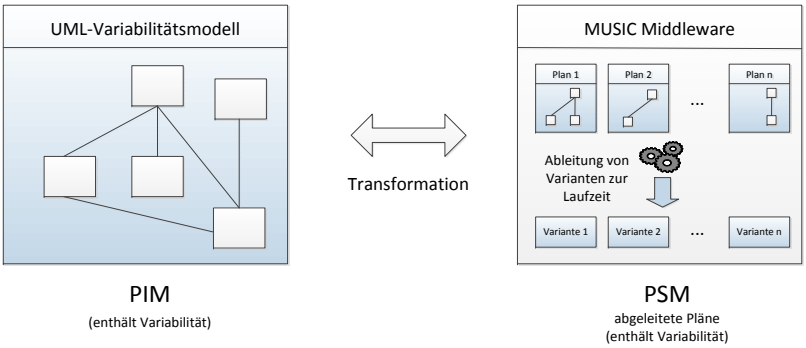

Abbildung 2: Der MDD-Prozess in MUSIC

Entwicklung ermöglicht das modellgetriebene Entwicklungsparadigma eine abstrahierte Vorgehensweise, indem anwendungsspezifische Design- und Implementierungsentscheidungen zunächst abstrakt in Modellen erfasst und nach und nach mit weiteren Details über die Anwendung ergänzt werden. Der Vorteil hier ist die starke Entkopplung von Design und Implementierung. Anwendungen können dadurch unabhängig von der verwendeten Plattform konstruiert werden. In der Entwurfsphase wird zuerst das plattformunabhängige Modell erstellt (engl. Platform Independent Model, PIM). Der Software-Architekt kann sich somit auf die Anwendung konzentrieren ohne mit detaillierten Implementierungsentscheidungen konfrontiert zu werden. Das PIM wird anschließend in ein plattformabhängiges Modell überführt (engl. Platform Specific Model, PSM). Bei dieser Transformation wird das PIM mit den nötigen plattformabhängigen Informationen angereichert. Das PSM kann bereits die Form von ausführbaren Programmcode haben.

Die Modelle werden also direkt zur Generierung von Quellcode genutzt, welcher anschließend ggf. noch von den Entwicklern anzupassen bzw. zu ergänzen ist.

In MUSIC werden aus dem Variabilitätsmodell die unterschiedlichen Pläne abgeleitet, die anschließend auf der Plan-basierten Middleware ausgeführt werden. Ein Plan ist hier eine in Java implementierte Spezifikation aus dieser anschließend eine oder mehrere Varianten abgeleitet werden (Abbildung 2). Die Umwandlung des UML-Modells in den Java-Programmcode führt ein Transformationsskript durch, welches die zusätzlich benötigten Informationen hinzufügt. Das Transformationsskript generiert das Grundgerüst der Anwendung - die eigentliche Funktionalität muss der Entwickler manuell hinzufügen. Der zweite Umwandlungsschritt von den Plänen zu den Ausführungsvarianten wird zur Laufzeit durch die Middleware durchgeführt. Bei einem signifikanten Kontextwechsel bewertet die Adaptions-Middleware jede dieser einzelnen Varianten anhand einer Nutzenfunktion. Diese Nutzenfunktion vergleicht die erforderlichen Eigenschaften der dabei relevanten Komponenten aller Varianten mit den Eigenschaften der aktuellen Situation, in der sich die Anwendung im Augenblick der Bewertung befindet. Die Variante, die die höchste Bewertung erhält, wird letztendlich ausgewählt und die Anwendung wird neu konfiguriert. Darüberhinaus werden im *Domänenmodell* Kontextinformationen, Kontextabhängigkeiten und Quality-of-Service(QoS)- Eigenschaften definiert. Diese sind notwendig, um zur Laufzeit die Varianten auszuwählen, die für die aktuelle Situation am besten geeignet sind. Um auch hoch dynamische heterogene Umgebungen zu unterstützen, werden diese in Form einer Ontologie modelliert, so dass beispielsweise auch verschiedene Repräsentation und Transformationen von Kontextdaten möglich sind.

Anforderungen an eine selbst-adaptive Anwendung müssen also dahingehend analysiert werden, dass 1) die verschiedenen Modi, in denen sich eine Anwendung befinden kann, erfasst werden und 2) alle benötigten Kontext- und Ressourcenabhängigkeiten identifiziert werden.

5 Vorgehen

Die explizite Berücksichtigung von adaptivem Verhalten im Softwareentwicklungsprozess erfordert einige Veränderungen. Die Identifizierung der verschiedenen Situationen, in denen sich eine Anwendung befinden kann, muss explizit berücksichtigt werden. Auch die verschiedenen Kontextinformationen, die in diesen Situationen benötigt werden, müssen herausgestellt werden, damit ein korrektes Adaptionsverhalten abgeleitet werden kann.

Soll Adaptionsverhalten, aufbauend auf der zuvor beschriebenen MUSIC-Vorgehensweise, im Vorfeld der Implementierung ermöglicht werden, so sind die entsprechenden Anforderungen frühzeitig zu erarbeiten. Die Notwendigkeit der einzelnen Adaptionen kann dabei über die Anforderungen aller Funktionen verteilt sein. Somit muss der Entwickler die Anforderungen unterscheiden können, die bereits für die Erstellung der Adaptionsmodelle benötigt werden, und den Anforderungen, die erst bei der konkreten Implementierung der Komponenten notwendig sind. Für die Modellierung der Adaption werden unterschiedliche Anforderungen benötigt.

Am Anfang der Entwicklung steht die generelle Anwendungsidee, die als Anwendungsszenario beschrieben werden kann. Die Anwendungsidee ergibt sich entweder aus einem konkreten Projektkontext oder wird beispielsweise durch einen Kunden vorgegeben. Durch die Szenarien wird der Einsatzbereich und das Ziel der Anwendung sehr schnell deutlich. Solche Szenarien

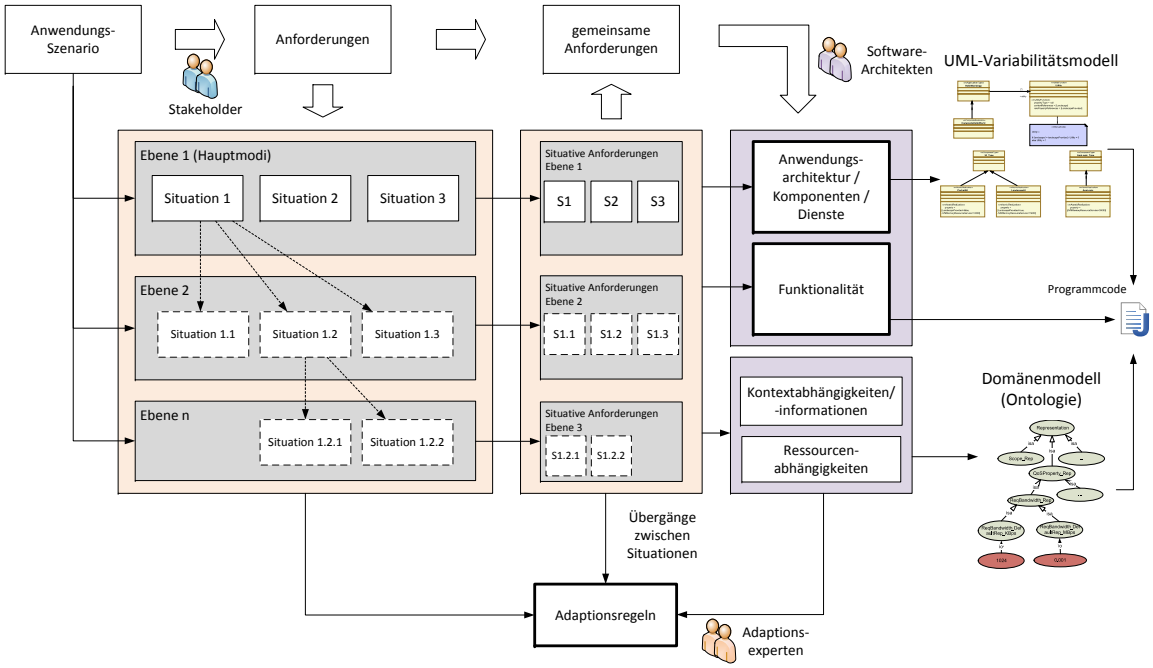

Abbildung 3: Methodisches Vorgehen

sind ein weit genutztes Mittel der Anforderungsanalyse [Poh08]. Mit Hilfe der Szenarien können Anforderungen an die Anwendung unter Einbeziehung aller Stakeholder effektiver erhoben und verständlicher kommuniziert werden.

Um die nötigen Informationen zum Adaptionsverhalten einer dynamischen Anwendung zu gewinnen, werden zu Beginn aus den Anwendungsszenarien die unterschiedlichen Situationen, in denen sich die Anwendung befinden kann, abgeleitet. Diese sind von großer Bedeutung, da in diesen Situationen die Anwendung gezwungen ist, auf unterschiedliche Implementierungen der Komponenten zurückzugreifen und diese einzubinden. Die Situationen stellen unterschiedliche Anforderungen an die Anwendung, die eventuell verschiedene Funktionen nötig machen, welche die Anwendung unter bestimmten Bedingungen bereitstellen muss (Siehe Abbildung 3). Die Situationen werden in Ebenen unterteilt. Situationen der folgenden Ebene verfeinern dabei die Situationen der vorherigen. So können Abhängigkeiten hinsichtlich einer Dimension als Ordnungsrahmen zur besseren Übersicht genutzt werden.

Die unterschiedlichen Situationen, in denen sich die Anwendung befinden kann, bringen eigene Anforderungen an die Anwendung mit. Diese situativen Anforderungen sind somit die Anforderungen, die sich in mindestens einer der Situationen von den anderen Situationen unterscheidet. Mit Hilfe der situativen Anforderungen lassen sich zwei Dinge ableiten. Zum Ersten kann mit ihnen eine sinnvolle Entscheidung über die zu modellierenden Komponenten der Anwendung getroffen werden. Dies orientiert sich an der Tatsache, dass unterschiedliche Anforderungen durch die Einbindung unterschiedlicher Komponenten oder Dienste (Adaption) gewährleistet werden kann. So ist es nicht zwingend zu wissen, wie im Detail die geänderte Anforde-

rung aussehen kann. Es muss aber bekannt sein, dass die Anforderungen sich in diesem Bereich ändern können. Zum Zweiten können mit Hilfe der Anforderung an die einzelnen Situationen bereits hier Ressourcenabhängigkeiten abgeleitet werden. Diese sind in späteren Schritten der Umsetzung von Bedeutung.

Die Situationen mit ihren Anforderungen bestimmen die späteren Varianten der Anwendungen. Damit die Anwendung sinnvoll zwischen den Varianten wechseln kann, werden für die Übergänge Kontextinformationen benötigt. Ändert sich also die Situation, in der sich der Nutzer befindet, ändert sich auch sein Kontext. Die Unterschiede des Kontext können somit aus den Änderungen der situativen Anforderungen abgeleitet werden. Aus diesen Kontextinformationen lassen sich die Adaptionsregeln für die Anwendung gewinnen. Mit diesen entscheidet später die Middleware über den Einsatz der unterschiedlichen Varianten.

Die Architektur der Anwendung mit ihren Komponenten wird bei MUSIC in einem Variabilitätsmodell modelliert. Kontextinformationen bestimmen das Domänenmodell, indem das benötigte Wissen über die Ausführungsumgebung(en) hinterlegt ist. Aus diesen beiden Modellen lässt sich dann die Anwendungsstruktur erstellen, in welcher dann die Funktionalität der Anwendung umgesetzt wird.

6 Fallbeispiel

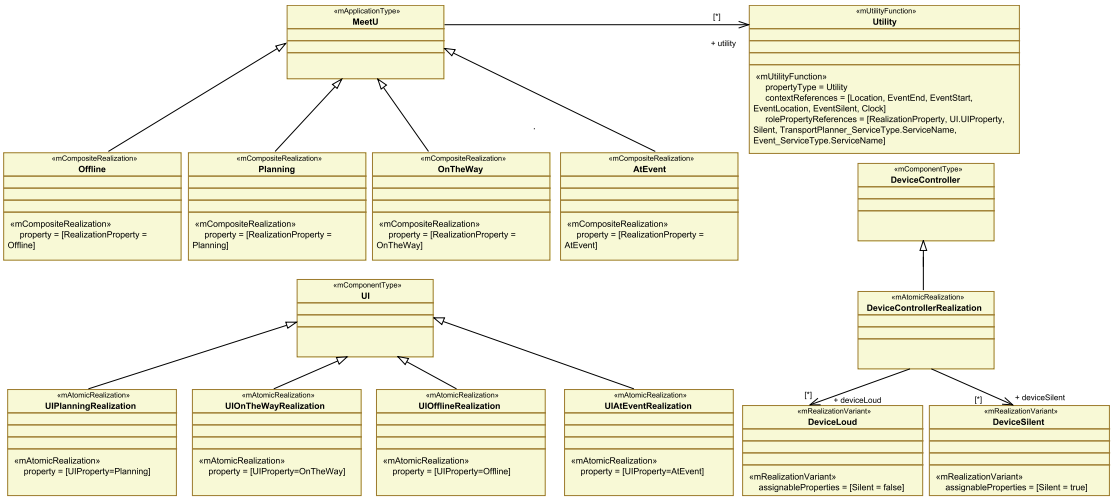

Abbildung 4: Ausschnitt aus dem MeetU-Variabilitätsmodell

Das im vorherigen Kapitel vorgestellte methodische Vorgehen soll anhand eines Fallbeispiels demonstriert werden. Dafür wird zunächst der Aufbau eines Szenarios beschrieben. „Meet-U“ soll eine mobile ubiquitäre Anwendung werden, mit welcher ein Benutzer die Möglichkeit hat, Verabredungen beziehungsweise Treffen jeder Art zu organisieren. Ein Treffen kann beispielsweise ein Konzert-, Theater- oder Arztbesuch mit Freunden oder Bekannten sein. Der Benutzer muss die Personen aber nicht zwangsläufig kennen. Mit Hilfe von persönlichen Präferenzen besteht die Möglichkeit Treffen mit unbekannten Personen zu organisieren. Die gewünschte An-

wendung soll den Benutzer in jeder Situation bis hin zum Treffen unterstützen. Wie im Vorgehen beschrieben, werden zunächst die charakteristischen Situationen der gewünschten Anwendung herausgestellt. Auf der ersten Ebene findet der Planungsprozess statt. Die zweite Ebene umfasst die Situationen, sobald das Treffen ansteht. Die gewünschte Anwendung soll adaptieren, um die Anreise beispielsweise durch eine Navigation zu unterstützen. Zuletzt erreicht der Anwender das Treffen (dritte Ebene). Hier sollen weitere Funktionen das Treffen unterstützen. Zum Beispiel könnte die Anwendung auf einem Konzert die Funktion bieten, Fotos mit Freunden auszutauschen.
Aus den vorgestellten Situationen ergeben sich folgende situative Anforderungen. „Planning“ beschäftigt sich mit den grundlegenden Aufgaben, die für die Organisation eines Treffens notwendig sind. So z.B. eine Kalenderfunktion, Freundeverwaltung, und Terminverwaltung. Die Situation „OnTheWay“ benötigt eine Funktionalität, um die Anreise zu organisieren. Auch an dieser Stelle sind unterschiedliche Realisierungen möglich. Die Anreise kann zu Fuß oder per Auto erfolgen. Unter Umständen soll der Anwender innerhalb eines Gebäudes navigiert werden oder auch zusätzliche Informationen während der Reise erhalten. „AtEvent“ soll in der MeetU-Anwendung die Möglichkeit bieten, Fotos beziehungsweise Daten unter Freunden zu tauschen oder lokale Dienstleistungen vor Ort dynamisch einbinden. Daher ergibt sich für die gemeinsamen Anforderungen eine Sammlung von Diensten, welche die gewünschten Funktionalitäten bereitstellen. Falls keiner dieser Dienste zur Verfügung steht, wechselt die Anwendung in die „Offline“ Situation, in der der Benutzer keine weiteren Aktionen durchführen kann.
Die Ressourcen können aus den beschrieben Situationen abgeleitet werden. Die Anwendung verwendet einen GPS- und einen Rfid-Sensor zur Positionsbestimmung. Diese Sensordaten werden als Kontextinformationen verwendet. Zudem nutzt die Anwendung die Termine der Treffen. Aus diesen Kontextinformationen können die Adaptionsregeln abgeleitet werden. Abhängig von der Zeit und dem aktuellen Ort findet der Wechsel zwischen den Situationen statt. Je nach Lautstärke des Umgebung passt sich die Anwendung aufgrund der Realisierungen „DeviceSilent“ beziehungsweise „DeviceLoud“ an. Abhängig von den verfügbaren Sensoren sowie der verbleibenden Akkukapazität nutzt die Anwendung GPS oder auch GSM zur Positionsbestimmung.
Die Anwendung bietet beliebige Erweiterungsmöglichkeiten. Abhängig von der Art des Treffens besteht die Möglichkeit, die vorgestellten Situationen beliebig zu gestalten. Das resultierende Variabilitätsmodell der Anwendung ist in Grafik 4 dargestellt.

7 Zusammenfassung

Der Artikel greift das auf Model-Driven-Development basierende Vorgehen MUSIC für die Entwicklung selbst-adaptiver Anwendungen auf und zeigt an diesem, wie Anforderungen für die Umsetzung der Adaptivität erkannt werden können. Mit Hilfe von Anwendungsszenarien werden Situationen abgeleitet, in denen sich die Anwendung befinden kann. Jede der Situationen stellt unterschiedliche Anforderungen an die Anwendung. Diese und die übergreifenden gemeinsamen Anforderungen bestimmen die Anwendungsarchitektur mit den Diensten und Komponenten, in denen die Funktionalität umgesetzt wird. Die Anforderungen der einzelnen Situationen legen Ressourcenabhängigkeiten fest. Zudem erlauben sie die benötigten Kontextinformationen für ein Adaptionsverhalten zu bestimmen. Mit ihnen werden in Kenntnis der Übergänge zwischen

den Situationen die Adaptionregeln festgelegt. In Zukunft soll diese Vorgehensweise erweitert werden, um Beschränkungen an die Softwarearchitektur bereits frühzeitig zu erkennen. Ebenfalls wünschenswert ist die Berücksichtigung rechtlicher Anforderungen direkt im Adaptionsverhalten einer Anwendung. Damit Anforderungen unabhängig von einem Rahmenwerk oder einer Middleware werden, ist eine weitere Möglichkeit, die erhobenen Anforderungen zunächst unabhängig in Anforderungsmodellen zu definieren.

Literatur

[BCZ05] D. M. Berry, B. H. C. Cheng, J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *In 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ*. P. 05. 2005.

[Che09] B. H. C. Cheng et al. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Cheng et al. (eds.), *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science 5525, pp. 1–26. Springer, 2009.

[FD96] A. Finkelstein, J. Dowell. A comedy of errors: the London Ambulance Service case study Software Specification and Design. In *The 8th International Workshop on*. Pp. 2 – 4. Schloss Velen, 1996.

[FP6] FP6 IST MUSIC Project. http://ist-music.berlios.de/.

[Gei09] K. Geihs et al. A comprehensive solution for application-level adaptation. *Softw. Pract. Exper.* 39:385–422, March 2009.

[GRWK09] K. Geihs, R. Reichle, M. Wagner, M. Khan. Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments. In Cheng et al. (eds.), *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science 5525, pp. 146–163. Springer Berlin / Heidelberg, 2009.

[Hal09] S. Hallsteinsen (ed.). *MUSIC Deliverable D13.06: MUSIC vision and solutions*. 2009.

[HBR02] T. Hall, S. Beecham, A. Rainer. Requirements problems in twelve software companies: an empirical analysis. *IEE Proceedings Software* 149:153 – 160, 2002.

[HHL10] A. Hoffmann, H. Hoffmann, J. M. Leimeister. Nutzerintegration in die Anforderungserhebung für Ubiquitous Computing Systeme. In *Workshop über Selbstorganisierende, adaptive, kontextsensitive verteilte Systeme (SAKS 2010). Electronic Communications of the EASST*. Volume 27, p. 9. Berlin, Germany, 2010. 188 (36-10).

[KD07] J. O. Kephart, R. Das. Achieving Self-Management via Utility Functions. *IEEE Internet Computing* 11:40–48, 2007.

[NE00] B. A. Nuseibeh, S. M. Easterbrook. Requirements Engineering: A Roadmap. In Finkelstein (ed.), *22nd International Conference on Software Engineering, ICSE'00*. IEEE Computer Society Press, 2000.

[Poh08] K. Pohl. *Requirements engineering: Grundlagen, Prinzipien, Techniken*. dpunkt-Verl., Heidelberg, 2. edition, 2008.

[Rou09] R. Rouvoy et. al. MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. In Cheng et al. (eds.), *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science 5525, pp. 164–182. Springer Berlin / Heidelberg, 2009.

[Wag10] M. Wagner (ed.). *MUSIC Deliverable D6.5: Modelling notation and software development method for adaptive applications in ubiquitous computing environments*. 2010.