Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit Anwendungen 1

Tobias Hutzler

Selbst-adaptive Software

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tobias Hutzler

Selbst-adaptive Software

Seminarausarbeitung im Rahmen der Veranstaltung Anwendungen 1
im Studiengang Informatik (Master of Science)
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Prüfer : Prof. Dr. rer. nat. Kai von Luck
Betreuer : Prof. Dr. Olaf Zukunft

Abgegeben am 25. Juli 2008

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Die zunehmende Leistungfähigkeit und Verbreitung mobiler Endgeräte, wie PDAs und Smartphones ermöglichen die Entwicklung verteilter und mobiler Softwaresysteme. Die begrenzten Ressourcen mobiler Endgeräte und die doch zunehmende, aber nicht vollständig verlässliche Netzinfratruktur erfordern die Anpassungsfähigkeit (Adaption - Kapitel 2) der Software an die aktuell gegebene Situation (Kontext - Kapitel 3) auf dem mobilen Endgerät.

1.2. Zielsetzung

Die vorliegende Arbeit will Grundlagen und Konzepte *selbst-adaptiver Software* vermitteln, sowie aktuelle und bereits bekannte Technologien für eine Umsetzung der Selbst-Adaption aufzeigen. Diese Arbeit geht auf vorhandene Problemstellungen, mögliche Ansätze und Lösungsstrategien ein. Darüber hinaus bietet diese Arbeit dem Autoren die nötige Grundlage, um im Projekt „Pervasive Gaming“ bei der Entwicklung eines Frameworks für mobile Anwendungen, Selbst-Adaption zu berücksichtugen und umzusetzen.

1.3. Inhaltlicher Aufbau

In Kapitel 2 wird das Thema dynamische Adaption behandelt. Ein weiterer Aspekt selbstadaptiver Software ist der aktuelle Kontext, in welchen sich die Software befindet und der Auslöser für Anpassungen am System ist. Dies ist unter dem Kaptiel 3 zu finden. In Kaptiel 4 wird auf offene Punkte und aktuellen Forschungesbedarf zum diesem Thema eingegangen. Abschließend werden im Ausblick die angestrebten Ziele dargestellt, welche der Autor weiter verfolgt.

1.4. Abgrenzung

Der Schwerpunkt dieser Arbeit wird auf dynamischer Adaption liegen. Multi-Agenten-Systeme, mit denen sich selbst-adaptive Anwendungen ebenfalls realisieren lassen, wird in dieser Arbeit als ein durchaus interessanter Ansatz erwähnt, aber nicht weiter aufgeführt. Abgrenzen möchte sich der Autor auch ganz bewusst von Arbeiten aus dem Bereich „Automatic Computing“ [KC03, ibm03], welches das Thema sehr viel weiter fasst und allgemeiner betrachtet. Einen Überblick und eine Zusammenfassung dieses Themas gibt Maik Weindorft in der Ausarbeitung [Wei06c] im Rahmen der Veranstaltung Anwendungen 1.

2. Adaption

Dynamische Adaption in ihren einfachsten Formen findet schon seit längere Zeit statt. Beispielsweise kann ein Laptop die Helligkeit des Bildschirms automatisch an die Lichtverhältnisse der Umgebung anpassen. Dies erfordert nicht zwingend eine durch Software gesteuerte Adaption. Konventionelle Regelkreise mit Helligkeitssensor und vordefinierten Stellwerten meistern diese Aufgabe. Für eine in Software geschmolzene Adaption kann als Beispiel das *Transmission Control Protocol* (TCP) genannt werden. TCP passt abhängig von der Last- und Fehlersituation im Netzwerk die Größe des Übertragungsfensters an [Ins81]. Diese Art der dynamischen Adaption wird *parametrische Adaption* genannt und kommt in vielen Formen in *kontextsensitiven Anwendungen* vor. Der nächste Abschnitt stellt die *parametrische Adaption* genauer dar, um den Leser den Unterschied zwischen dieser und der *kompositionellen Adaption* zu verdeutlichen.

2.1. Parametrische Adaption

Der Zustand der Umgebung beeinflusst interne Parameter der Anweundung und führt zu einem adaptiven Verhalten [MSKC04]. Parametrische Adaption verändert nicht die Struktur einer Anwendung, sondern nur einzelne Stellgrößen, welche in Abhängigkeit von beobachteten Werten das Verhalten der Anwendung steuern. Der Entwickler legt zur Entwurfszeit die Adaptionsmöglichkeiten der Anwendung durch entsprechende Parameter fest. Daraus ergeben sich einige Einschränkungen dieser Adaptionstechnik. So ist es beispielsweise nicht möglich zur Laufzeit ganze Komponenten der Anwendung auszutauschen. Daher kann diese Adaptionsart als „Feintuning“ der Anwendung verstanden werden.

2.2. Kompositionelle Adaption

> *Compositional adaptation results in the exchange of algorithmic or structural parts of the system, in order to improve a program's fitness to its current environment* [MSKC04].

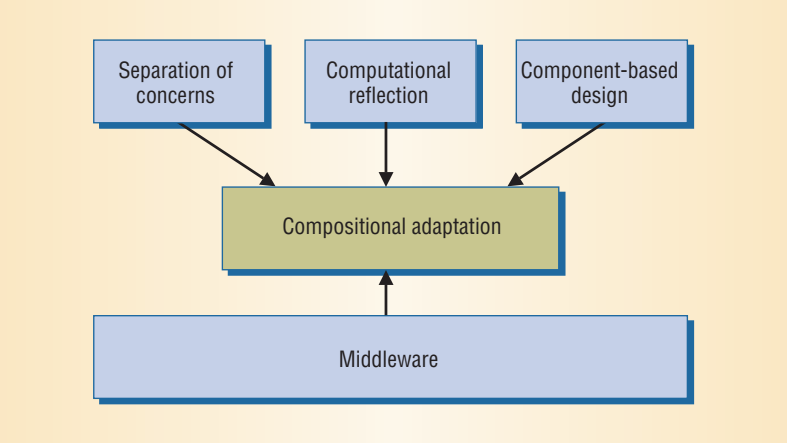


Abbildung 2.1.: Schlüsseltechnologien für kompositionelle Adaption [MSKC04]

Kompositionelle Adaption ist eine weitaus mächtigere Form der Softwareadaption [MSKC04]. Sie ermöglicht den kontextabhängigen Austausch von Komponenten zur Laufzeit. Dadurch kann sich die ganze Anwendungsstruktur des Softwaresystems verändern, um sich ihrer aktuellen Umgebung anzupassen. Man bezeichnet diese Form der Adaption deshalb auch als architekturbasierte Adaption (vgl. [CHG+04]). Im Vergleich zur rein parametrischen Adaption bietet diese Form der Adaption eine weitaus höhere Flexiblität und gibt dem Entwickler den nötigen Gestaltungsfreiraum, um aktuelle und zukünfitge pervasive und mobile Anwendungsszenarien umzusetzen.
Allerdings ist anzumerken, dass die Realisierung der Anwendungen sehr viel schwieriger wird. Auf der Hand liegt, dass dafür eine entsprechende Systeminfrastruktur und eine hochgradig modulare Anwendungsstruktur gegeben sein muss, welche dynamisches Binden unterstützt. Dies ist für den Austausch, die Migration und die Instanziierung von Komponenten nötig. Im folgenden Abschnitt werden einige vorhandene Technologien vorgestellt, die genau dies ermöglichen.

2.3. Vorhandene Technologien

Allgemeiner Ausgang vieler Ansätze für eine *kompositionelle Adaption* ist ein notwendiger Grad von Indirektion, welche Überwachung (*Interception*) und Neuausrichtung (*Redirection*) an Codestellen in der Anwendung ermöglicht (vgl. [MSKC04]). Die Realisierung *selbstadaptiver Software* wurde durch die Konfluenz der drei Schlüsseltechnologien: *Separation of Concerns*, *Reflections* und *Component-Based Design* (siehe 2.1) gestützt. Im Folgenden werden diese Technologien genauer erläutert.

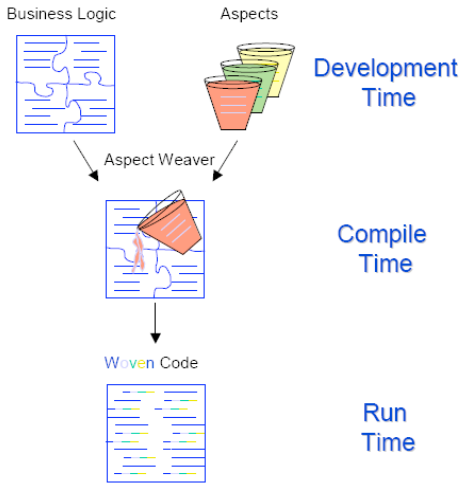


Abbildung 2.2.: Konzeptionelle Darstellung von Aspect Weaving [MSKC04]

2.3.1. Separation of Concerns

Separation of Concerns [HW01, TO01] ermöglicht die getrennte Entwicklung funktionalen Verhaltens (Business Logic) von *Crosscutting Concerns* (querschnittliche Belange) [1]. *Separation of Concerns* ist zu einem anerkannten und weit verbreiteten Prinzip in der Softwareentwicklung geworden, und wird in verschiedenen Entwicklungstechniken mit unterschiedlichem Grad umgesetzt. Neben *domain-specific languages, generative programming, generic programming, constraint languages*, ist *Aspektorientierte Programmierung (AOP)* [KLM+97] einer der bekanntersten und verbreitetsten Ansätze. Abbildung 2.2 stellt das Aspect Weaving dar. Die Codeimplementierungen der *Crosscutting Concerns* werden als *Aspects* bezeichnet, welche als eigenständige Teile eines Systems an bestimmten Programmstellen, den *Pointcuts* durch eine *Aspect-Weaver* zu Compilierungszeit oder zu Laufzeit in den Anwendungscode eingewebt werden. Die Anwendung kann dadurch ein neues Verhalten erlangen. AOP unterstützt dynamische Rekomposition in drei Wegen (vgl. [MSKC04]):

- Viele Anpassungen des Systems korrelieren mit den Crosscutting Concerns (z.B. Sicherheit, Energiemanagement, Ausfallsicherheit).
- Aspect-Weaving zur Compilierungszeit, ermöglicht nur eine eingeschränke Adaptionsunterstützung. Wird das Aspect-Weaving zur Laufzeit verlagert, erschließt sich ein systematischer Lösungsansatz, um die dynamsische Rekomposition zu gewährleisten.

[1] Als Crosscutting Concerns (CCC) werden querschnittliche Belange einer Software bezeichnet, deren Modularisierung besonders in objektorientieren Progammiersprachen oft nicht oder nur schwer realisierbar sind. Im Allgemeinen handelt es sich hierbei um nichtfunktionale Anforderungen, wie beispielsweise Sicherheitsaspekte oder eben Adaption, die in der Anwendung quer verstreut realisiert werden müssen.

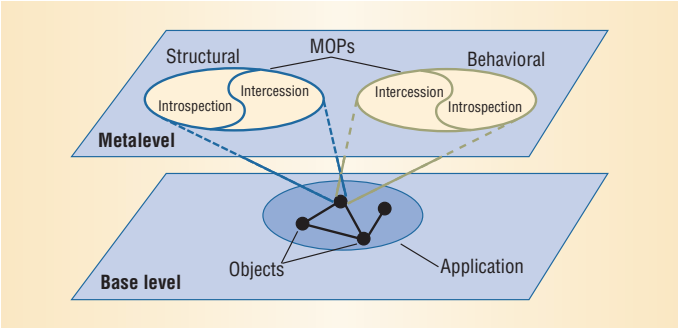


Abbildung 2.3.: Meta-level Zusammenhänge werden in Meta-object Protokollen (MOPs) gesammelt [MSKC04]

- Wird Adaption als ein „genereller“ Aspekt modelliert, ist es möglich, dass zur Entwurfszeit noch nicht existierende Adaptionsentscheidungen erreicht werden können.

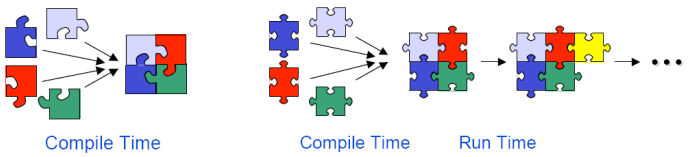
2.3.2. Reflections

Reflections bezieht sich auf die Fähigkeit eines Programms sich selbst zu beobachten und darauf zu agieren und möglicherweise das eigene Verhalten anzupassen [Mae87]. Durch Abstraktion kann ein selbstbeschreibendes Modell eines Programms erstellt werden. Das selbstbeobachtende Verhalten einer Anwendung wird als *Introspection* bezeichnet. Während man die Reaktion auf das Beobachtete als *Intercession* bezeichnet.

Abbildung 2.3 stellt ein reflektives System (*base-level objects*) und seine Selbstrepräsentation (*meta-level objects*) dar. Beide Modelle sind kausal voneinander abhängig, was bedeutet: Änderungen an einem Modell wirken sich direkt auf das andere Modell aus. Ein Meta-Objekt-Modell (MOP) ermöglicht die systematische (im Gegensatz zur ad hoc) *Introspection* und *Intercession* der Anwendung. Darüber hinaus kann das MOP in *Structural* [2] oder *Behavioral* [3] *Reflection* kategorisiert werden. Eine selbst-adaptive Software kann Reflections dazu nützen, sich selbst zu beobachten und darauf zu reagieren. In Kombination mit AOP können in den Reflections des MOPs die Logik für das Einweben (Aspect Weaving) der *Crosscutting Concerns* zur Laufzeit untergebracht werden.

[2] Structural Refelctions, beinhalten die Klassenhierarchie, Klasseninterkonnektivität und Datentypen. So kann beispielsweise festgestellt werden, welche Methoden ein bestimmtes Objekt versteht.

[3] Behavioral reflection treffen eine Aussage über die berechenbare Semantik einer Anwendung. So kann beispielsweise eine verteilte Anwendung über die Auswahl und Verwendung eines am besten geeigneten Protokolls für die aktuelle Situation entscheiden.



(a) Static recomposition. (b) Dynamic recomposition.

Abbildung 2.4.: Statische und Dynamsiche Rekomposition auf komponeten-basiertem Design [MSKC04]

2.3.3. Komponenten-basiertes Design

Ein komponenten-basiertes Design ist nötig aber nicht hinreichend für eine kompositionelle Adaption. Sie bildet die Grundlage für die dynamsiche Rekomposition. Wohldefinierte Interfaces spezifizieren die Schnittstelle zwischen Dienst-Anbieter (service provider) und Dienst-Nutzer (service client). Die Spezifizierung dient als Vertrag an den sich die beteiligten Parteien halten müssen und ermöglicht dadurch eine unabhängige Implementierung dieser (Design by contract). Abbildung 2.4(a) zeigt eine statische Rekomposition zur Compilierungszeit, deren Ergebnis die resultierende Anwendung ist. In Abbildung 2.4(b) ist die dynamische Rekomposition zur Laufzeit dargestellt. Es werden zur Laufzeit Komponenten ausgetauscht, wodurch sich die Anwendungsstruktur und somit unter Umständen auch die Architektur der Anwendung verändern kann.

2.4. Klassifizierung von kompositioneller Adaption

Forscher und Entwickler haben bereits eine Fülle von Anwendungen, Middelware und Techniken bereitgestellt, welche kompositionelle Adaption zu einem bestimmten Grad unterstützen. Mc Kinley et. al. haben eine Klassifizierung dieser Ansätze vorgenommen. Sie nehmen dabei eine Klassifizierung nach Art (wie?), Zeit (wann?) und Ort (wo?) kompositionelle Adaption greift, vor. Im Folgenden werden diese Dimensionen erläutert. Eine Liste der untersuchten Anwendungen/Projekte A.1 bzw. Techniken A.2 kann der Leser im Anhang finden. Mc Kinley weist ausdrücklich darauf hin, dass diese Listen nicht vollständig sind, und darüber hinaus noch weitere existieren.

2.4.1. Wie komponieren?

Die erste Dimension beschäftigt sich mit speziellen Softwaretechniken, die kompositionelle Adaption ermöglichen. Abbildung A.2 listet spezielle Softwaretechniken mit einer Kurzbe-

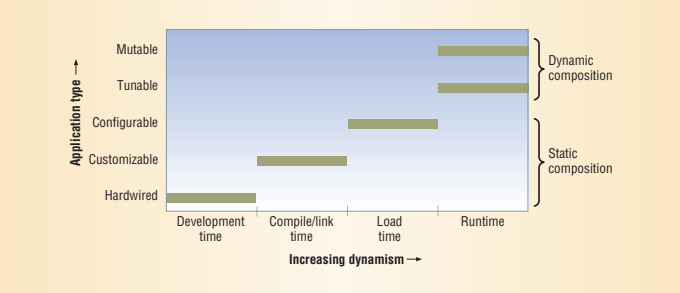


Abbildung 2.5.: Klassifikation der Softwarekomposition nach Kompositions- bzw. Rekompositionszeit [MSKC04]

schreibung und Einsatzbeispielen auf. Eine Diskussion findet in [AC03] von Mehmet Aksit und Zièd Choukair statt (siehe auch [HM07]). Alle Techniken haben gemeinsam, dass sie Indirektion und Interaction verwenden. Um dies zu erreichen, kommen in einigen Ansätzen bestimmte Entwurfsmuster zum Einsatz. Anderen nützen AOP oder Reflections oder beide in Kombination, um die kompositionelle Adaption zu erreichen (siehe [MRC07]).

2.4.2. Wann komponieren?

Die zweite Dimension beschäftigt sich mit der Frage, wann Komposition stattfinden soll. Allgemein lässt sich folgende Aussage treffen: Spätere Bindung ermöglicht eine mächtigere Art der Adaption, allerdings erschwert es auch die Sicherstellung von Konsistenz des adaptiven Softwaresystems. Wird beispielsweise die Komposition der Adaption zur Entwurfs- oder Entwicklungszeit oder Compilierungszeit festgelegt, wird die Dynamik eingeschränkt. Alledings ist es einfacher, sicherzustellen, dass kein annomales Verhalten der Adaption auftreten kann. Findet die Komposition andererseits zur Laufzeit statt, so kann eine höhere Dynamik der Adaption garantiert werden. Diese Anwendungen auf Korrektheit nach traditionellen Testmethoden und Verifikationsmethoden zu überprüfen, gestalltet sich dadurch weitaus schwerer. Bei Applikationstypen, deren Bindung zur Entwicklungs-, Compilierungs- oder Ladezeit stattfinden, spricht man auch von *statischer Komposition*, während Applikationstypen, welche zur Laufzeit die Bindung durchführen, von *dynamischer Komposition* spricht (siehe Abbildung 2.5).

2.4.3. Wo komponieren?

Die zuletzt untersuchte Dimension beschäftigt sich mit der Frage, wo die kompositionelle Adaption stattfinden soll. Auch hier existieren unterschiedliche Ansätze. Die Adaption

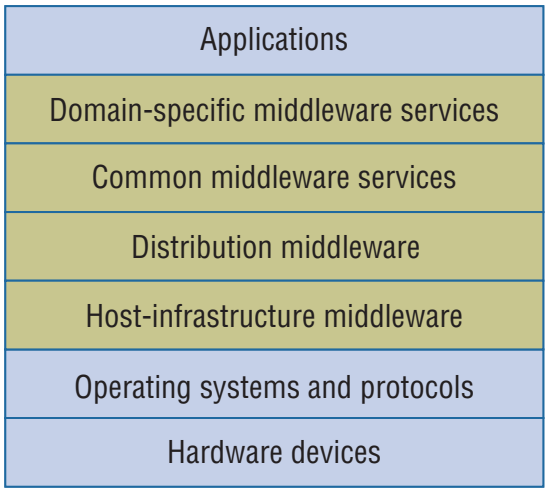


Abbildung 2.6.: Vier-Ebenen Dekomposition der Middelware nach D. C. Schmidt [Sch02]

kann einerseits direkt in der Applikation oder in einer der Middelware-Layer nach D. C. Schmidt (siehe Abbildung 2.6) stattfinden. Mc Kinley hat die Untersuchung der Adaption auf Betriebssystem-Ebene nicht weiter untersucht.

2.5. Adaptionsentscheidung

Ein weiterer wichtiger Punkt im Zusammenhang mit selbst-adaptiver Software ist die Automatisierung der Adaptionsentscheidung, die eine besondere Herausforderung darstellt. Wie kann man feststellen, welche Komponenten unter einer Anzahl von vielen Alternativen diejenige ist, welche die aktuelle Stituation am besten erfüllt? Hier kommen eine Reihe von unterschiedlichen Techniken zum Einsatz, welche im nächsten Abschnitt vorgestellt werden.

2.5.1. Regel-basierte Adaptionsentscheidung

In regelbasierten Systemen steuert eine Menge vorgegebener Regeln das Anpassungsverhalten [LJK+00]. Beispielsweise könnte eine solche Regel wie folgt lauten:
IF (ResponseTime>100msec) THEN (increase CPU by 5 %)
Problem an den Situations-Aktions Regeln ist, dass sie sich nur bedingt für dynamische Adation eigenen. Einersteits steigt bei größeren Adaptionsmöglichkeiten die Komplexität immens. Andereseits müssen alle Möglichkeiten zur Entwurfszeit feststehen, wodurch sich keine dynamische Adapiton zur Laufzeit realisieren lässt (vgl. [Gei08, MSKC04]).

2.5.2. Ziel-orientierte Adaptionsentscheidung

Weitaus geschickter ist die Idee, ein zu erreichendes Ziel vorzugeben und die betroffenen Kompontenten selbst entscheiden zu lassen, was getan werden muss, um diese Ziel zu erreichen. Das oben aufgeführte Beispiel mit der Responsetime würde sich in einem ziel-orientierten System wie folgt formulieren lassen: *ResponseTime< 100ms*.
Dem System bleibt es selbst überlassen das vorgegebene Ziel zu erreichen. Die Entscheidungsfindung wird dadurch zum Planungsproblem, was in die Disziplin „Künstliche Intelligenz“ der Informatik fällt.
Viel Arbeit wird aktuell in das Gebiet der Multi-Agenten-Systeme gesteckt. In einigen Multi-Agenten-Systemen werden die Agenten als autonom, ziel-orientiert beschrieben, die die Fähigkeit haben, mit ihrer Umwelt und untereinander zu kommunizieren. Durch die Kooperation der individuellen Agenten wird ein Gesamtziel verfolgt [TCW+04, Flo07].

2.5.3. Nutzen-orientierte Adaptionsentscheidung

Aktuell werden auch Methoden aus der künstlichen Intellligenz [RN03] und ökonomisch ausgerichtete Optimierungsansätze erprobt [KD07, PSGS04, WTKD04].
Ein probates Mittel ist die Maximierung einer Nutzenfunktion, die den Nutzen einzelner Realisierungsvarianten nummerisch auf einer Skala abbildet. Mit geeignet gewählten Nutzenfunktionen lassen sich ganz unterschiedliche Aspekte der Adaption abbilden. Beispielsweise können Aussagen über Kosten einer Kommunikationsverbindung oder der Ressourcenverbrauch unterschiedlicher Komponenten getroffen werden. Die Adaptionsentscheidung basiert auf einer vom Entwickler vorgegebenen Nutzenfunktion, welche diejenige Konfiguration des Systems oder Anwendung auswählt, die für den aktuellen Kontext den größten Nutzen bringt (vgl. [Gei08]). Die Adaptionsentscheidung wird zum Optimierungsproblem.

2.6. Konfiguration

Für die Umsetzung der Adaptionsentscheidung ist das Konfigurationsmanagement zuständig. Rekonfiguration kann dazu führen, dass Komponenten instantiiert, ersetzt, gelöscht oder verlagert werden. Dies kann sich auch direkt auf die Beziehung der Komponenten zueinander auswirken. Die laufende Anwendung muss unterbrochen werden und in adaptierer Form wieder gestartet werden. Eine Rekonfiguration muss die Konsistenz der Komponetenbeziehung sicherstellen und ggf. den Zustand von zustandsbehafteten Komponenten von der alten auf die neue Konfiguration übertragen. Da Zustandsrepräsentationen anwendungsspezifisch sind und beliebig komplex sein können, ist eine generische Lösung für den Zustandstransfer nicht möglich (vgl. [Gei08]).

3. Kontext

Geeigente Definitionen für den Begriff Kontext sind in der Literatur an vielen Stellen zu finden. Die Definition nach Dey scheint nach aktuellem Zeitgeist sehr passend.

> *„Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves“* [Dey00].

3.1. Zusammenhang zwischen selbst-adaptiver Software und Kontext

Adaptive Anwendungen sind kontextsenitiv [Gei08], was bedeutet, das Adaption immer eine Reaktion auf eine Änderung der Software selbst oder der Umgebung, in der sie exisitert, ist. Kontextsensoren liefern die notwendigen Rohdaten, um aus Änderungen des Kontexts zu schlussfolgern. Aus einer Reihe von Klassifikationen von Kontextinformationen wird in dieser Arbeit auf die Klassifikation nach Schilit [SAW94], bei der nach *Ressourcen-*, *Umgebungs-* und *Benutzerkontext* unterschieden wird, zurückgegriffen.

3.1.1. Ressourcenkontext

Der Ressourcenkontext umfasst die Ressourcen der Rechner, auf denen die Anwendungen ausgeführt werden. Beispiele dafür sind verfügbare Speicherkapazität, Kommunikationsbandbreite, Energiezustand, CPU-Auslastung oder Interaktionsschnittstellen. Meist sind diese Informationen betriebssystemintern vorhanden und erfordern dadurch keiner zusätzlichen Hardware-Sensoren. Der Zugang über Schnittstellen zu den Ressourceninformationen weicht allerdings besonders bei mobilen Endgeräten ab, und wird nicht immer bereitgestellt (vgl. [Gei08]).

3.1.2. Umgebungskontext

Der Umgebungskontext ist eine Reflektion des Zustandes der Umwelt, in der die Anwendung abläuft. Ein bereits sehr intensiv erforschte und oft benutzte Form der Kontextinformation ist die Ortsangabe, die beispielsweise über GPS-Sensoren gewonnen werden kann. Darüber hinaus existieren Kontextinformationen über Geräuschpegel, Temparatur, Licht oder die Existenz bestimmter Objekte im Umfeld, welche dem Umgebungskontext zugeordnet werden können. Hierfür sind spezielle Hardwaresensoren erforderlich, um die Parameter zu überwachen. Man kann davon ausgehen, dass in zunehmenden Maße Sensoren für die GPS-Lokalisierung in mobilen Endgeräten zur Verfügung stehen. Sensorenkomponenten, die zur Erfassung der anderen Umgebungsinformationen dienen, sind noch nicht Teil der Standardausrüstung mobiler Rechner, können aber über Standardschnittstellen angeschlossen werden (vgl. [Gei08]).

3.1.3. Benutzerkontext

Die zuletzt hier aufgeführte Form, die für Adaptionsentscheidungen relevant sein kann, ist der Benutzerkontext. Hierzu zählen die aktuellen Vorlieben des Benutzers, längerfristige verwendbare Benutzerprofile und Zusatzinformationen wie Terminkalender oder Kontaktinformationen.
Die automatische Erfassung relevanter Benutzerkontextinformationen gestaltet sich weitaus schwieriger als die beiden vorher genannten Kontextarten (vgl. [Gei08]).

Kontextrohdaten ermöglichen die Schlußfolgerung auf Situation wie Konzertbesuch, Vorlesungsveranstaltung oder Konferenzteilnahmen. Dies ist eine anspruchsvolle, aber machbare Aufgabe, wie beispielsweise die Anwendung „locale“[1] gezeigt hat.

3.2. Verwandte Arbeiten

Kontextsensitive Anwendungen sind aktuelles Forschungsthema. Interessierten Lesern sind weitere Themen, die an der HAW Hamburg im Rahmen der Veranstaltungen Anwendungen 1 und 2 sowie in den Projektarbeiten abgehandelt wurden, zu empfehlen (siehe [Ulr07, Wei06b, Wei06a]).

[1] Eine Anwendung, die in Abhängigkeit der Position und der Kalendereinträge auf die aktuelle Situation schließt und daraufhin das Mobiltelefonprofil einstellt. Entwickelt von Studenten des M.I.T. im Rahmen der Android Developer Challenge von Goolgle(http://code.google.com/android/adc.html referenziert am 18.05.2008)

4. Resümee

4.1. Fazit und offene Fragen

Projiziert man die gegenwärtig technische Entwicklung auf dem Gebiet der dynamischen Adaption in die Zukunft, so zeichnet sich unmittelbar ab, dass dynamische Adaption ein hervorstechendes Attribut von Software sein wird. Nötig ist anpassungsfähige, robuste und erweiterbare Software, die trotz dynamsicher schwankender Kontexteigenschaften ihren Nutzen behält. Das Einsatzgebiet selbst-adatpiver Software ist sicher nicht nur auf den Bereich „Mobile Computing“ oder „Pervasive Computing“ beschränkt. „Embedded Systems“ stellen ganz ähnliche Anforderungen.
Selbst-Adaptivität bleibt und ist aktuelles Forschungsgebiet und eine Reihe von Forschungsprojekten haben sich dieser Herausfordungen bereits angenommen. Die Machbarkeit selbstadaptiver Softwaresysteme bleibt außer Frage, allerdings haben sich durch bisherige Forschungsanstrengungen neuer Problemstellungen ergeben, deren Lösung unabdingbar sind. Um nur die wichtigsten offenen Fragen zu nennen, ist sicherlich an erster Stelle die Benutzbarkeit erwähnenswert. Ist es gewollt, dass sich die Software selbst adaptiert? Wie wird die Selbst-Adaption durch den Benutzer empfunden? Wie oft darf sich eine Software anpassen, ohne dass dies als störend empfunden wird? Diese Fragestellungen eröffnen weitere Problemfelder: Wie kann man besonders in mobilen Anwendungen Selbst-Adaption validieren und testen? Kann eine zuverlässige Adaption garantiert werden? Können kritische Aktionen nachgewiesen werden? Dadurch kommen auch Fragestellungen zur Sicherheit ins Spiel: Ist ein System, das sich ständig ändert, vertrauenswürdig? Wie kann man die potenziellen persönlichen Kontextinformationen der Benutzer schützen? Diese Reihe von Fragen verlangt noch viel Forschungsarbeit und wird teilweise erst durch den alltäglichen Einsatz selbstadaptiver Software beantwortet werden können (vgl. [Gei08]).

4.2. Ausblick

Der Autor wird im kommenden Semester im Rahmen des Projekts „Pervasive Gaming“ an der Entwicklung einer geeigneten Framework-Architektur für mobile Anwendungen mitwirken. Besonders in Hinblick auf die Integration und Umsetzung eines selbst-adaptiven Systems

können die gewonnenen Erkenntnisse dieser Ausarbeit für Konzeption und prototypische Realisierung nutzender Applikationen eingesetzt werden. Die im Rahmen dieser Ausarbeitung gewonnen Erkenntnise bilden eine sehr gute theoretische Grundlage. Der Autor kann sich weiterhin vorstellen, aus diesem Themengebiet ein geeignetes Thema (siehe 4.1) für die anstehende Masterthesis zu wählen. Zu diesem Zeitpunkt eine konkrete Entscheidung zu treffen, ist allerdings noch zu früh.

Abbildungsverzeichnis

Literaturverzeichnis

[AC03] AKSIT, MEHMET und ZIÈD CHOUKAIR: *Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision*. In: *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, Seite 84, Washington, DC, USA, 2003. IEEE Computer Society.

[CHG+04] CHENG, SHANG-WEN, AN-CHENG HUANG, DAVID GARLAN, BRADLEY SCHMERL und PETER STEENKISTE: *Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure*. icac, 00:276–277, 2004.

[Dey00] DEY, ANIND KUMAR: *Providing architectural support for building context-aware applications*. Doktorarbeit, Atlanta, GA, USA, 2000. Director-Gregory D. Abowd.

[Flo07] FLOCH, JACQUELINE: *D2.2 Theory of adaptation*, 2007. http://www.intermedia.uio.no/display/madam/Home , Dec 2006, referenziert 18.05.2008.

[Gei08] GEIHS, KURT: *Selbst-Adaptive Software*, April 2008.

[HM07] HOLGER MÜGGE, TOBIAS RHO, DANIEL SPEICHER PASCAL BIHLER ARMIN B. CREMERS: *Programming for Context-based Adaptability — Lessons learned about OOP, SOA, and AOP*. Workshop on Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS.07), in proceedings of Kommunikation in verteilten Systemen (KiVS.07), February 26 - March 2, 2007, Bern, Schweiz, Mar 2007.

[HW01] HOFFMAN, DANIEL M. und DAVID M. WEISS (Herausgeber): *Software fundamentals: collected papers by David L. Parnas*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[ibm03] *An architectural blueprint for autonomic computing*. 2003. http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf , 21.07.08.

[Ins81] INSTITUTE, INFORMATION SCIENCES: *RFC 793*, 1981. Edited by Jon Postel. Available at http://rfc.sunsite.dk/rfc/rfc793.html.

[KC03] KEPHART, JEFFREY O. und DAVID M. CHESS: *The Vision of Autonomic Computing*. Computer, 36(1):41–50, January 2003.

[KD07] KEPHART, JEFFREY O. und RAJARSHI DAS: *Achieving Self-Management via Utility Functions*. IEEE Internet Computing, 11(1):40–48, 2007.

[KLM+97] KICZALES, GREGOR, JOHN LAMPING, ANURAG MENHDHEKAR, CHRIS MAEDA, CRISTINA LOPES, JEAN-MARC LOINGTIER und JOHN IRWIN: *Aspect-Oriented Programming*. In: AKŞIT, MEHMET und SATOSHI MATSUOKA (Herausgeber): *Proceedings European Conference on Object-Oriented Programming*, Band 1241, Seiten 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[LJK+00] LI, B., W. JEON, W. KALTER, K. NAHRSTEDT und J. SEO: *Adaptive Middleware Architecture for a Distributed Omnidirectional Visual Tracking System*, 2000.

[Mae87] MAES, PATTIE: *Concepts and experiments in computational reflection*. SIGPLAN Not., 22(12):147–155, 1987.

[MRC07] MÜGGE, HOLGER, TOBIAS RHO und ARMIN B. CREMERS: *Integrating aspect-orientation and structural annotations to support adaptive middleware*. In: *MAI '07: Proceedings of the 1st workshop on Middleware-application interaction*, Seiten 9–14, New York, NY, USA, 2007. ACM.

[MSKC04] MCKINLEY, PHILIP K., SEYED MASOUD SADJADI, ERIC P. KASTEN und BETTY H. C. CHENG: *Composing Adaptive Software*. Computer, 37(7):56–64, 2004.

[PSGS04] POLADIAN, VAHE, JOAO PEDRO SOUSA, DAVID GARLAN und MARY SHAW: *Dynamic Configuration of Resource-Aware Services*. In: *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, Seiten 604–613, Washington, DC, USA, 2004. IEEE Computer Society.

[RN03] RUSSELL, STUART J. und PETER NORVIG: *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[SAW94] SCHILIT, BILL, NORMAN ADAMS und ROY WANT: *Context-Aware Computing Applications*. In: *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[Sch02] SCHMIDT, DOUGLAS C.: *Middleware for real-time and embedded systems*. Commun. ACM, 45(6):43–48, 2002.

[TCW+04] TESAURO, GERALD, DAVID M. CHESS, WILLIAM E. WALSH, RAJARSHI DAS, ALLA SEGAL, IAN WHALLEY, JEFFREY O. KEPHART und STEVE R. WHITE: *A Multi-Agent Systems Approach to Autonomic Computing*. In: *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Seiten 464–471, Washington, DC, USA, 2004. IEEE Computer Society.

[TO01] TARR, PERI und HAROLD OSSHER: *Workshop on advanced separation of concerns in software engineering*. In: *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, Seiten 778–779, Washington, DC, USA, 2001. IEEE Computer Society.

[Ulr07] ULRICH, JAROWSLA: *Context-Aware Systeme: aktuelle Projekte*, 2007. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/urich/bericht.pdf, 21.07.2008.

[Wei06a] WEINDORF, MAIK: *Autonomic Computing*, 2006. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07-aw/weindorf/report.pdf, 21.07.2008.

[Wei06b] WEINDORF, MAIK: *Context-Aware Mobile Computing*, 2006. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07/weindorf/report.pdf, 21.07.2008.

[Wei06c] WEINDORF, MAIK: *Self-Managing & Context-Aware Mobile Computing*, 2006. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2006/weindorf/abstract.pdf, 21.07.2008.

[WTKD04] WALSH, WILLIAM E., GERALD TESAURO, JEFFREY O. KEPHART und RAJARSHI DAS: *Utility Functions in Autonomic Systems*. icac, 00:70–77, 2004.

A. Projekte und Techniken, die kompositionelle Adaption unterstützen

Table 1. Example research projects, commercial packages, and standard specifications that provide compositional adaptation.

Project	Institution/Organization
Language-based projects	
AspectJ	Xerox Palo Alto Research Center
Composition filters	Universiteit Twente, The Netherlands
Program Control Language (PCL)	University of Illinois
Open Java	IBM Research
R-Java	University Federal de São Carlos, Brazil
Kava	University of Newcastle, UK
Adaptive Java	Michigan State University
Transparent Reflective Aspect Programming in Java (TRAP/J)	Michigan State University
Middleware-based projects	
Domain-specific services layer:	
Boeing Bold Stroke (BBS)	Boeing
Common services layer:	
CorbaServices	Object Management Group
Quality objects (QuO)	BBN Technologies
Adaptive Corba Template (ACT)	Michigan State University
Interoperable Replication Logic (IRL)	University of Rome, Italy
Distribution layer:	
.NET remoting	Microsoft
Open ORB and Open COM	Lancaster University, UK
The ACE ORB (TAO) and Component Integrated ACE ORB (CIAO)	Distributed Object Computing Group
DynamicTAO and Universally Interoperable Core (UIC)	University of Illinois
Orbix, Orbix/E, and ORBacus	Iona Technologies
Squirrel	University of Kaiserslautern, Germany
AspectIX	Friedrich-Alexander University, Germany
Host infrastructure layer:	
Java virtual machine (JVM)	Sun Microsystems
Common Language Runtime (CLR)	Microsoft
Iguana/J	Trinity College, Dublin
Prose	Swiss Federal Institute of Technology
Adaptive Communication Environment (ACE)	Distributed Object Computing Group
Ensemble	Cornell University
Cross-layer projects	
Distributed Extensible Open Systems (DEOS)	Georgia Institute of Technology
Grace	University of Illinois

Abbildung A.1.: Beispiele für Projekte, kommerzielle Anwendungen und Spezifikationen, die kompositionelle Adaption unterstützen [MSKC04]

Table 2. Software recomposition techniques.

Technique	Description	Examples
Function pointers	Application execution path is dynamically redirected through modification of function pointers.	Vtables in COM, delegates and events in .NET, callback functions in Corba
Wrappers	Objects are subclassed or encapsulated by other objects (wrappers), enabling the wrapper to control method execution.	ACE, R-Java, PCL, QuO, TRAP/J
Proxies	Surrogates (proxies) are used in place of objects, enabling the surrogate to redirect method calls to different object implementations.	ACT, AspectIX
Strategy pattern	Each algorithm implementation is encapsulated, enabling transparent replacement of one implementation with another.	DynamicTAO and UIC
Virtual component pattern	Component placeholders (virtual components) are inserted into the object graph and replaced as needed during program execution.	ACE and TAO
Metaobject protocol	Mechanisms supporting intercession and introspection enable modification of program behavior.	Open Java, Kava, TRAP/J, Open ORB, Open COM, Iguana/J
Aspect weaving	Code fragments (aspects) that implement a crosscutting concern are woven into an application dynamically.	AspectJ, Composition Filters, TRAP/J, AspectIX, Iguana/J, Prose
Middleware interception	Method calls and responses passing through a middleware layer are intercepted and redirected.	ACT, IRL, Prose
Integrated middleware	An application makes explicit calls to adaptive services provided by a middleware layer.	Adaptive Java, Orbix, Orbix/E, ORBacus, BBS, CIAO, Iguana/J, Ensemble

Abbildung A.2.: Software-Rekompositionstechniken [MSKC04]