1 Einleitung

Adaptive Anwendungen können sich auf Grund von Änderungen ihrer Umgebung dynamisch rekonfigurieren, u.a. um von den Nutzern vorgegebene Anwendungseigenschaften (*quality of service*) zu gewährleisten, die durch die variierende Verfügbarkeit von Ressourcen unterliegender Infrastrukturen verletzt werden [Kon 2000]. Das Phänomen der variierenden Verfügbarkeit von Ressourcen wird nach [Dini u. a. 2004] als *partial ressources* bezeichnet und kann z.B. durch die Verwendung mobiler Kommunikationsnetze verursacht werden, die durch physikalische Gegebenheiten wie die Entfernung eines Nutzers zu einer Basisstation oder die Anzahl von Nutzern in einer Funkzelle variierende Übertragungsbandbreiten oder Fehlerraten besitzen. Auch die Nutzung des Internets über wechselnde Kommunikationsverbindungen wie drahtgebundene und drahtlose Netzwerke verursachen Änderungen in den verfügbaren Ressourcen, auf die adaptive Anwendungen entsprechend reagieren müssen. Eine dynamische Rekonfiguration kann dabei einfache Parameteranpassungen, die Aktualisierung von Anwendungslogik während der Laufzeit oder Änderungen der Softwarearchitektur beinhalten.
Neben der Verfügbarkeit von Ressourcen können Änderungen der Umgebung auch geänderte Anforderungen durch einen Nutzer oder das Bekanntwerden von Sicherheitslücken umfassen. In diesem Fall ist die dynamische Rekonfiguration eine wichtige Voraussetzung für die Erhöhung der Verfügbarkeit einer Anwendung durch die Möglichkeit des Einspielens aktualisierter Softwareversionen (*hot-fix*). Wird die Rekonfiguration während der Laufzeit durchgeführt, können Ausfallzeiten durch einen Neustart vermieden werden. Während das Einspielen von aktualisierten Softwareversionen eine Maßnahme zur Fehlervermeidung darstellt, kann die dynamische Rekonfiguration auch eingesetzt werden, um bei einem Ausfall von Teilen einer Anwendung, Fehler zu beheben [Laprie 1998].

1.1 Konfiguration verteilter komponentenbasierter Anwendungen

Verteilte Anwendungen werden auf einer Reihe von Rechnern ausgeführt, die über ein Netzwerk miteinander verbunden sind (verteiltes System). Für die Entwicklung von Software hat in den letzten Jahren der Ansatz der Objektorientierung eine weite Verbreitung gefunden. In der Objektorientierung wird die betrachtete Welt in Objekte mit ihren Eigenschaften und Operationen gekapselt. Die Struktur eines Objekts wird durch seine Attribute (Eigenschaften) beschrieben. Das Verhalten eines Objekts wird durch seine Methoden (Operationen) festgelegt. Eine **objektorientierte verteilte Anwendung** besteht aus einer Reihe von Objekten, die auf über ein Netzwerk verbundenen Rechnern existieren und über entfernte Methodenaufrufe (*Object Remote Procedure Call* (ORPC)) miteinander kommunizieren.
Eine wichtige Technologie zur Strukturierung komplexer Software ist die Verwendung von Softwarekomponenten. In Analogie zu Hardwarebausteinen werden Softwarekomponenten als austauschbare, unabhängig erworbene Elemente der Wiederverwendung

aufgefasst. Nach [Szyperski 1999, S. 36 ff.] ist eine Softwarekomponente eine ausführbare Einheit unabhängiger Herstellung, Beschaffung und Deployment, die durch Dritte in ein funktionierendes System komponiert werden kann. Unter **Deployment** wird in der Informatik die Auslieferung und Installation von Software in einem verteilten System verstanden [Object Management Group 2006, S. 1]. Eine Softwarekomponente besitzt wohl definierte Schnittstellen, über die auf ihre Funktionalität zugegriffen werden kann. Softwarekomponenten sind eine Hülle für zusammenhängende Software in binärer Form und enthalten Vorlagen für die Laufzeitstrukturen einer Anwendung. Eine **komponentenbasierte Anwendung** wird aus einer Reihe von Softwarekomponenten erzeugt, die im Falle der Objektorientierung Vorlagen für die Objekte einer Anwendung enthalten.

Die Komponentenorientierung ist dabei nicht nur technischer Natur, sondern hat auch einen wirtschaftlichen Hintergrund. Durch die Schaffung von durch Konkurrenz belebte Märkte können die Qualität der aus Komponenten zusammengesetzten Endprodukte gesteigert sowie die Entwicklungskosten reduziert werden [Szyperski 1999, S. 4 ff.]. Aus der Definition nach Szyperski folgt, dass eine Komponentenart (Menge von Komponenten gleicher Schnittstelle) von mehreren unabhängigen Herstellern erstellt werden kann und dabei mit unterschiedlichen Qualitätsmerkmalen bzw. Eigenschaften vorliegen kann. Ein Anwendungsentwickler kann bei der Komposition einer komponentenbasierten Anwendung aus einer Menge funktionsgleicher Komponenten wählen, um z.B. die Einhaltung eines bestimmten finanziellen Budgets einzuplanen, indem entsprechende Komponenten ausgewählt werden. Ein wichtiges Ziel der Komponentenorientierung ist es, einen Markt alternativer Komponenten zu erstellen.

Genau an dieser Stelle sind die Konzepte der Komponentenorientierung für die Entwicklung adaptiver Software von zentraler Bedeutung; Komponenten sind unabhängige Einheiten, die nach ihrer Komposition Systeme mit unterschiedlichen Eigenschaften ergeben, die sich in bestimmten Umgebungen besser eignen als andere Alternativen. Mit der Auswahl entsprechender Komponenten bei der Komposition einer Anwendung kann also die Anwendung an die Eigenschaften ihrer Umgebung angepasst werden, indem die besten Alternativen aus einer Reihe funktionsgleicher Komponenten ausgewählt werden.

Szyperski definiert den Begriff der Softwarekomponente als Hülle zusammenhängender Software in binärer Form. In den verbreiteten Java- und .NET-Komponentenplattformen werden Komponenten während der Anwendungslaufzeit durch Objekte zum Leben erweckt und die Kapselung, die zunächst logisch durch die Hülle der Komponenten geschaffen wurde, geht verloren. Für die Entwicklung adaptiver Software ist es nun wichtig, die funktionale Kapselung von Komponenten auf die Laufzeit von Anwendungen zu erweitern. Im Rahmen dieser Arbeit fasst eine Komponente während der Laufzeit die Menge aller Objekte zusammen, deren Klassen bzw. Objektprototypen in einer Deployment-Einheit „Softwarekomponente“ im Sinne der Szyperski-Definition enthalten sind. Die Austauschbarkeit von Softwarekomponenten kann so auf die Laufzeit einer Anwendung ausgeweitet werden. Um die Komponentenabstraktion während der Laufzeit zu differenzieren wird der Begriff der **Kapsel** eingeführt, die die beschriebene Zusammenfassung einer Objektmenge während der Laufzeit realisiert.

Komponentenbasierte, verteilte Anwendungen können als gerichtete Graphen mit Kapseln als Knoten und Verbindungen als Kanten betrachtet werden [Kramer und Magee 1990]. Kapseln werden auf Rechnern in einem verteilten System ausgeführt und interagieren in Form von Objekten über ihre Schnittstellen. Das Verhalten ein-

zelner Kapseln kann über Parameter konfiguriert werden. Die Verbindung zwischen den Objekten verschiedener Kapseln wird über Konnektoren realisiert, die als Interaktionsmedium dienen, über das entfernte Methodenaufrufe ausgeführt werden. Eine gültige Zusammenstellung parametrierter Kapseln, Konnektoren und einer Zuordnung der Kapseln auf die Rechner eines verteilten Systems wird als **Anwendungskonfiguration** bezeichnet. Der Konfigurationsbegriff wird in Kapitel 3 formal eingeführt.
Eine Anwendungskonfiguration geht aus einer Komposition von Komponenten aus einer Menge verfügbarer Alternativen hervor. Oft existieren verschiedene Möglichkeiten, aus einer Menge von Komponenten ein System zur Lösung einer bestimmten Aufgabe zu komponieren. Abbildung 1.1 zeigt die Auswahl von Komponenten und die Zusammenstellung zu gültigen Anwendungskonfigurationen. Die Komposition von Anwendungen wird nur am Rande dieser Arbeit diskutiert. Lösungsansätze wurden u.a. im Rahmen von Forschungsprojekten zum Thema „Ressourcenorientiertes Konfigurieren“ [Heinrich 1993] (einem Teilgebiet der Künstlichen Intelligenz) und seit kurzer Zeit im Zusammenhang mit der Komposition von Web-Diensten [Milanovic und Malek 2004] untersucht, sind aber auch Gegenstand aktueller Forschungsarbeiten [Richling 2006].

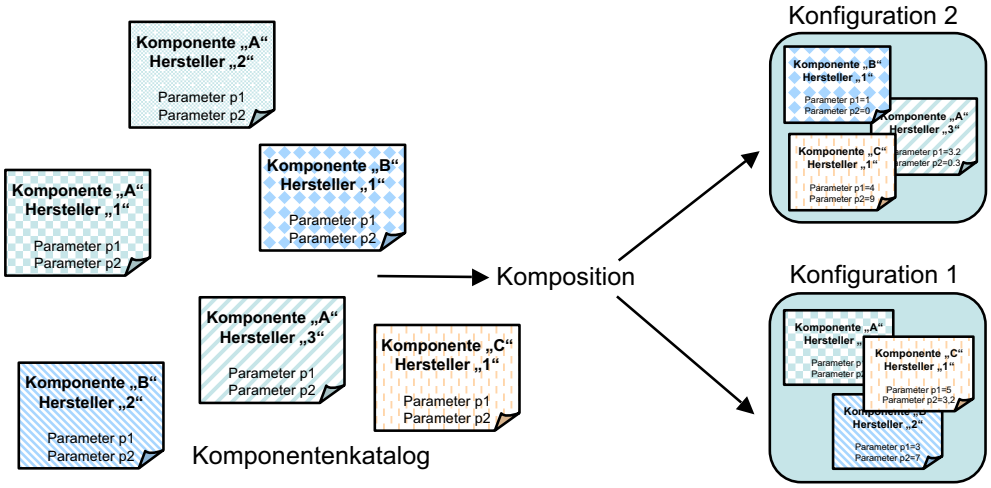

Abbildung 1.1: Komposition: Erstellung alternativer Konfigurationen

Neben den verfügbaren Ressourcen bestimmt die Anwendungskonfiguration einer verteilten komponentenbasierten Anwendung wesentlich ihre Eigenschaften, die adaptive Anwendungen in einem durch den Nutzer definierten Bereich halten sollen. Ändern sich die einer Anwendung zur Verfügung stehenden Ressourcen, können durch die **Auswahl einer alternativen Anwendungskonfiguration**, die für die gegebene Verfügbarkeit von Ressourcen optimiert ist, die geforderten Eigenschaften gewährleistet werden. Die Auswahl und Aktivierung einer neuen Anwendungskonfiguration während der Laufzeit wird als dynamische Rekonfiguration bezeichnet.
Die Verwendung der Konfigurationsabstraktion bietet eine hohe Flexibilität bei der Realisierung des adaptiven Verhaltens. Anpassungen an einer verteilten Anwendung können auf einer breiten Ebene vorgenommen werden. Angefangen bei der Parameteranpassung, die das Verhalten auf der lokalen Ebene einer Kapsel verändert, über den Einsatz zusätzlicher Kapseln (z.B. einer Datenkompressionskapsel zwischen zwei über ein Netzwerk kommunizierende Kapseln) bis hin zur Migration der Objekte einer Kapsel auf einen anderen Rechner, falls auf diesem z.B. mehr Rechenleistung zur Verfügung steht, existieren vielfältige Möglichkeiten.

1.2 Eine Ausführungsumgebung für adaptive Anwendungen

Ein wichtiges **Ziel dieser Arbeit** war die Entwicklung und Integration von Techniken zur Ausführung adaptiver komponentenbasierter verteilter Anwendungen in einer Komponentenplattform. Weit verbreitete Vertreter aktueller Komponentenplattformen sind Suns Java-Plattform und Microsofts .NET Plattform. Eine Besonderheit dieser Plattformen ist der Einsatz einer virtuellen Maschine für die Ausführung von objektorientierten Anwendungen, um die durch den Einsatz verschiedener Betriebssysteme und Hardware-Plattformen entstandene Heterogenität der Rechner in einem verteilten System zu verbergen, aber auch um die Portabilität von Software zu erhöhen. Die virtuellen Maschinen führen eine portable Zwischenrepräsentation (*Bytecode, Intermediate Language*) der Softwarekomponenten aus, mit der plattformspezifische Details wie z.B. der Zugriff auf Register und den Stack oder die Verwaltung von dynamischem Speicher gekapselt wird. Durch die Verwendung einer virtuellen Maschine für eine spezifische Plattform kann die Zwischenrepräsentation einer Softwarekomponente ohne Neuübersetzung des Quellcodes ausgeführt werden, indem die virtuelle Maschine die Zwischenrepräsentation direkt vor der Ausführung in Maschineninstruktionen (*just-in-time compilation*) übersetzt.

Bei der Integration von Mechanismen zur Ausführung adaptiver Anwendungen in eine Komponentenplattform ist es zum einen möglich, existierende virtuelle Maschinen um notwendige Mechanismen zu erweitern; im Wesentlichen um die fehlende Rekonfigurationsfunktionalität direkt zu integrieren. Dieser Ansatz ist problematisch, da erweiterte Versionen der virtuellen Maschinen dem Aktualisierungsprozess des Herstellers angepasst werden müssen, das heißt, dass die Änderungen in eine neue Version eingepflegt werden müssen. Zusätzlich kann die Stabilität der virtuellen Maschine gefährdet und damit der Einsatz in Produktivumgebungen erschwert werden. Deshalb wurden im Rahmen dieser Arbeit Techniken entwickelt, um die dynamische Rekonfiguration in den Java- und .NET-Plattformen zu ermöglichen, ohne die virtuelle Maschine zu erweitern.

Hierfür wurden existierende **Basistechnologien** für die Integration verwendet. Dies ermöglicht den herstellerunabhängigen Einsatz der entwickelten Techniken. Eine wichtige Basistechnologie und Grundlage der im Rahmen dieser Arbeit entwickelten Techniken ist die Verwendung der vorgestellten virtuellen Maschinen für die Ausführung von Softwarekomponenten in einem heterogenen verteilten System. Des Weiteren wird die Möglichkeit des dynamischen Ladens von Komponenten verwendet, um einer Konfiguration während der Laufzeit Objekte neuer Komponenten hinzufügen zu können. Beim Laden der Funktionalität einer neuen Komponente muss zunächst das Deployment, also die Verteilung und Installation, von Komponenten in einem verteilten System untersucht werden, bei dem die entsprechenden Komponenten und ihre Abhängigkeiten an einen entfernten Rechner übertragen werden müssen. Die Basistechnologie der Metaprogrammierung wird zur Manipulation von Objekten einer Anwendung aus einer Metaebene verwendet, über welche die Konfiguration der Anwendung erfolgt.

Eine weitere verwendete Basistechnologie ist die aspektorientierte Programmierung [Kiczales u. a. 1997], mit der nichtfunktionale Aspekte separiert von der eigentlichen Anwendungslogik entwickelt werden können. Im Rahmen dieser Arbeit wurde konfigurationsspezifische Logik identifiziert und als Aspekt implementiert. Mit Hilfe eines Aspektwebers kann diese Logik in die verwendeten Softwarekomponenten integriert werden und muss damit nicht vom Hersteller der Softwarekomponenten entwickelt

werden. Die entwickelten Aspekte ermöglichen damit die Wiederverwendbarkeit der konfigurationsspezifischen Logik bei der Erstellung adaptiver Anwendungen. Eine komplexe Anwendung fand die aspektorientierte Programmierung im Rahmen dieser Arbeit durch die Integration von Synchronisationslogik in die Anwendungskomponenten, die bei der dynamischen Rekonfiguration benötigt wird.

1.3 Dynamische Rekonfiguration

Der Wechsel von einer Anwendungskonfiguration zu einer anderen während der Laufzeit einer Anwendung wird als dynamische Rekonfiguration bezeichnet. Bei der dynamischen Rekonfiguration können verschiedene Konfigurationsänderungen unterschieden werden:

- Anpassung von Kapselparametern (z.B. Kompressionsrate)
- Hinzufügen/Entfernen von Kapseln
- Anpassung von Verbindungen zwischen Kapseln
- Austausch der Algorithmen einer Kapsel (Dynamische Aktualisierung)
- Änderung der Platzierung von Kapseln auf andere Rechner (Migration)

Eine dynamische Rekonfiguration kann komplexe Änderungen an der Struktur und der Logik einer Anwendung bedeuten, welche die Konsistenz der Daten und die Funktionalität der Anwendung insgesamt nicht beeinträchtigen dürfen. Hierfür müssen geeignete Algorithmen verwendet werden, um zu verhindern, dass Anwendungsaktivitäten auf Grund nicht abgeschlossener Rekonfigurationsoperationen Inkonsistenzen verursachen. Für die dynamische Rekonfiguration komponentenbasierter Anwendungen existieren Algorithmen [Bidan u. a. 1998; Kramer und Magee 1990; Moazami-Goudarzi 1999; Wermelinger 1997], die auf dem „Einfrieren“ der Anwendungen in einem konsistenten Zustand basieren. Dieser Zustand wird im Rahmen dieser Arbeit als rekonfigurierbarer Zustand bezeichnet.
Übertragen auf die Java- und .NET-Komponentenplattformen bedeutet das Erreichen eines rekonfigurierbaren Zustands, dass keine offenen Methodenaufrufe in den Objekten der betroffenen Kapseln existieren. Dies ist vor allem dadurch bedingt, dass ohne eine Manipulation der virtuellen Maschine kein Zugriff auf den Stack und die Register möglich ist, die potentiell Zustand in Form von lokalen Variablen enthalten können. Dies ist bei der dynamischen Aktualisierung problematisch, da auf dem Stack Referenzen auf Objekte existieren können, die aktualisiert wurden. Ein wichtiges Ziel bei der dynamischen Rekonfiguration ist das Erkennen eines rekonfigurierbaren Zustands, in dem kein Thread Instruktionen einer Methode der Objekte einer von der Rekonfiguration betroffenen Kapsel ausführt. Da dieser Zustand nicht immer erreicht werden kann, weil immer wieder eine neue Methode aufgerufen werden kann, während ein vorheriger Methodenaufruf noch verarbeitet wird, müssen neue Methodenaufrufe blockiert und die Beendigung aktiver Methodenaufrufe abgewartet werden, um einen rekonfigurierbaren Zustand herbeizuführen.
Im Falle verschachtelter Aufrufe muss beim Blockieren eine Reihenfolge beachtet werden, da sonst potentiell nicht alle aktiven Methodenaufrufe beendet werden können, weil diese auf die Ausführung verschachtelter Aufrufe warten. Im Falle azyklischer

Verbindungsgraphen kann das Erreichen des rekonfigurierbaren Zustands durch das geordnete Blockieren von Konnektoren implementiert werden [Wermelinger 1997], bei dem auf das Ende aktiver Methodenaufe über diesen Konnektor gewartet wird und neue Aufrufe blockiert werden. In diesem Zusammenhang untersucht die vorliegende Arbeit Techniken, die notwendige Synchronisationslogik automatisiert in die Komponenten der Anwendung zu integrieren. Mit Hilfe der aspektorientierten Programmierung konnte so die Synchronisationslogik wiederverwendbar für die Entwicklung adaptiver Anwendungen zur Verfügung gestellt werden und somit die Entwicklung vereinfacht werden.

Es hat sich herausgestellt, dass viele der existierenden Rekonfigurationsalgorithmen nicht für die Integration in eine Komponentenplattform wie der Java- bzw. .NET-Plattform verwendet werden können, weil diese Verfahren keine Unterstützung multipler Threads oder re-entranter Kapseln bieten. Re-entrante Kapseln können zyklische Aufrufbeziehungen von verschachtelten Methodenaufrufen verursachen, die beim Erreichen des rekonfigurierbaren Zustands problematisch sind. Eine zyklische Aufrufbeziehung entsteht, wenn ein Thread aus dem Kontext einer Methode weitere Methoden aufruft und dabei gleichzeitig zwei laufende Methodenrufe in einer Kapsel besitzt. Dies kann genau dann entstehen wenn im Verbindungsgraphen einer Anwendungskonfiguration Zyklen existieren. In diesem Fall kann für das Blockieren von Verbindungen keine Reihenfolge ermittelt werden, um Verklemmungen zu vermeiden.

Für Anwendungskonfigurationen mit zyklischen Verbindungsgraphen wurde deshalb im Rahmen dieser Arbeit ein neuer Algorithmus entwickelt, bei dem selektiv einzelne Methodenaufrufe blockiert werden. Der entwickelte Algorithmus basiert auf threadspezifischen Datenstrukturen, die in jeder Kapsel bei Ein- und Austritt eines Threads aktualisiert werden. Bei einer dynamischen Rekonfiguration kann für jeden Thread ermittelt werden, ob dieser aktive Methodenaufrufe an einer von der Rekonfiguration betroffenen Kapseln besitzt und damit neue Methoden aufrufen darf; andernfalls kann der Thread blockiert werden. Durch den Einsatz logischer Thread-IDs, die Threads über Rechnergrenzen hinweg eindeutig identifizieren, unterstützt das entwickelte Verfahren auch die Rekonfiguration verteilter Anwendungen. Eine besondere Eigenschaft des neuen Rekonfigurationsverfahren ist, dass die notwendige Synchronisationslogik nur geringe Zusatzkosten (in Form von zusätzlichen Instruktionen) für normale Methodenaufrufe verursacht.

Ein Spezialfall der dynamischen Rekonfiguration stellt die **dynamische Aktualisierung** von Kapseln dar. In diesem Fall liegen die entsprechenden Softwarekomponenten in einer neuen Version vor oder alternative Implementierungen sollen ohne einen Neustart der Anwendung aktiviert werden. Bei der dynamischen Aktualisierung ergibt sich dabei das Problem des Zustandstransfers zwischen alter und neuer Version. Existierende Ansätze realisieren dynamische Aktualisierungen durch den Einsatz redundanter Hardware oder erweiterter Laufzeitumgebungen [Dmitriev 2001]. Auch der Einsatz der Serialisierungsfunktionalität aktueller Komponentenplattformen für den Zustandstransfer ist nicht möglich, da Versionsinformationen Teil der persistierten Daten sind und bei der Deserialisierung auf die Softwarekomponenten der alten Version zugegriffen wird. Im Rahmen dieser Arbeit wurde ein Verfahren entwickelt, das die komplexe Rekonfigurationsoperation „Dynamische Aktualisierung“ ohne die Manipulation der Laufzeitumgebung ermöglicht. Durch die Verwendung der Metaprogrammierung (Reflexion) kann durch das Traversieren aller Objekte einer Kapsel erkannt werden, welche Objekte durch neue Versionen ersetzt werden müssen. Der Zustand der alten Objekte

wird durch attributweises Kopieren auf die neuen Objekte, die aus der aktualisierten Softwarekomponente erzeugt werden, übertragen.
Während die dynamische Rekonfiguration in modernen Komponentenplattformen den zentralen Fokus dieser Arbeit darstellt, wurden einige weiterführende Aspekte untersucht, die für die Funktionsweise und Entwicklung adaptiver Anwendungen von Bedeutung sind.

1.4 Funktionsweise adaptiver Anwendungen

Adaptive Anwendungen müssen in der Lage sein, Änderungen der Umgebungseigenschaften und ihres internen Zustands zu erkennen und entsprechende Rekonfigurationsmaßnahmen durchzuführen. Die Diagnose von Änderungen kann entweder durch periodische Messung der Umgebungseigenschaften und der internen Struktur oder ereignisbasiert durch entsprechende Mechanismen der unterliegenden Infrastruktur erfolgen. Die Umgebungseigenschaften können dabei die verfügbaren Ressourcen aber auch weitere Parameter wie den Kontext eines Nutzers (z.B. seine Position) umfassen. Abbildung 1.2 zeigt die Funktionsweise einer adaptiven Anwendung unter Verwendung eines Regelkreises. Ausgehend von der periodischen Messung der Umgebungseigenschaften und dem Zustand der Anwendung in einer Analysephase werden Einstellungen an der Anwendung vorgenommen.

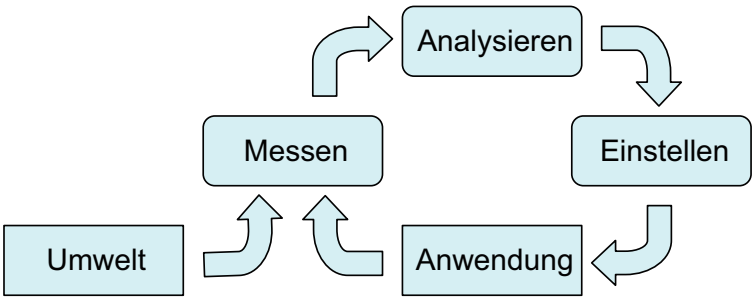

Abbildung 1.2: Zyklus einer adaptiven Anwendung

Die wesentlichen Grundmechanismen für die Realisierung adaptiver Anwendungen lassen sich in drei Bereiche unterteilen. Zunächst müssen die Umgebungseigenschaften ermittelt werden. Diese Funktionalität wird als **Monitoring** bezeichnet. Die Anpassung der Anwendung erfolgt durch die Auswahl einer geeigneten Anwendungskonfiguration und einer dynamischen Rekonfiguration der Anwendung. Schlussendlich muss die **Zuordnung** zwischen ermittelten Umgebungseigenschaften und den alternativen Konfigurationen beschrieben werden.
Neben der Implementierung einer Ausführungsplattform für adaptive Anwendungen, die das Deployment von Softwarekomponenten in verteilten Systemen, das Monitoring von Umgebungseigenschaften und Kapselausfällen sowie der Integration heterogener Java- und .NET-Kapseln untersucht, wurde im Rahmen der Arbeit ein graphisches Werkzeug zur Erstellung von Anwendungskonfigurationen sowie der Konfiguration des Monitoring und der Zuordnung von Konfiguration zu gemessenen Umgebungseigenschaften entwickelt. Das graphische Werkzeug unterstützt dabei auch die Entwicklung alternativer Konfigurationen mit Hilfe von Architekturmustern, mit denen auf ausgewiesene Änderungen der Umgebung reagiert werden kann. Die aspektorientierte

Programmierung spielt bei der Werkzeugunterstützung eine zentrale Rolle, denn sie ermöglicht, dass komplexe (re-)konfigurationsspezifische Logik generiert werden kann, und erleichtert damit die Entwicklung adaptiver Anwendungen. Hierfür wurde ein neues Programmiermodell für die Identifikation von Kommunikationsendpunkten und Parametern in den Komponenten entwickelt, die bei der graphischen Konfigurationserstellung für die Verbindung und Parametrierung von Kapseln verwendet werden.
Der entwickelte Ansatz zur Entwicklung adaptiver Anwendung wurde im Rahmen des **Distributed Control Lab** (DCL), einer entfernten Laborumgebebung für die Ausführung von Steuerungsexperimenten über das Internet, für die sichere Ausführung von studentischen Steuererungsprogrammen eingesetzt. Im DCL werden über einen Web-Browser eingegebene Programme, die Algorithmen für die Steuerung einer physikalischen Hardware enthalten, auf einem eingebetteten Rechner installiert und ausgeführt. Die über das Internet abgesendeten, potentiell fehlerhaften bzw. bösartigen Programme werden durch die entwickelte Ausführungsumgebung während der Laufzeit überwacht und beim Erkennen fehlerhaften Verhaltens oder dem Ausfall einer Nutzersteuerung (z.B. durch eine unbehandelte strukturierte Ausnahme) durch eine dynamische Rekonfiguration der Steuerungsanwendung ersetzt.

1.5 Struktur der Arbeit

Im Rahmen dieses Abschnitts wird die Struktur der vorliegenden Arbeit und der Zusammenhang der einzelnen Projekte, die im Rahmen der Arbeit beschrieben werden, vorgestellt. Eine Übersicht ist in Abbildung 1.3 dargestellt. Im Mittelpunkt der Arbeit steht dabei die Entwicklung adaptiver Anwendungen durch die dynamische Rekonfiguration komponentenbasierter Anwendungen, für die alternative Anwendungskonfigurationen für verschiedene Umgebungssituationen vorliegen. Die Grundpfeiler der Arbeit stellen Algorithmen zur dynamischen Rekonfiguration und dem Spezialfall der dynamischen Aktualisierung dar, die in den zentralen Kapiteln 3 und 4 beschrieben werden. Basierend auf den entwickelten Techniken wurde eine komplexe Fallstudie im Rahmen des Distributed Control Lab realisiert, die im 6. Kapitel der Arbeit vorgestellt wird.

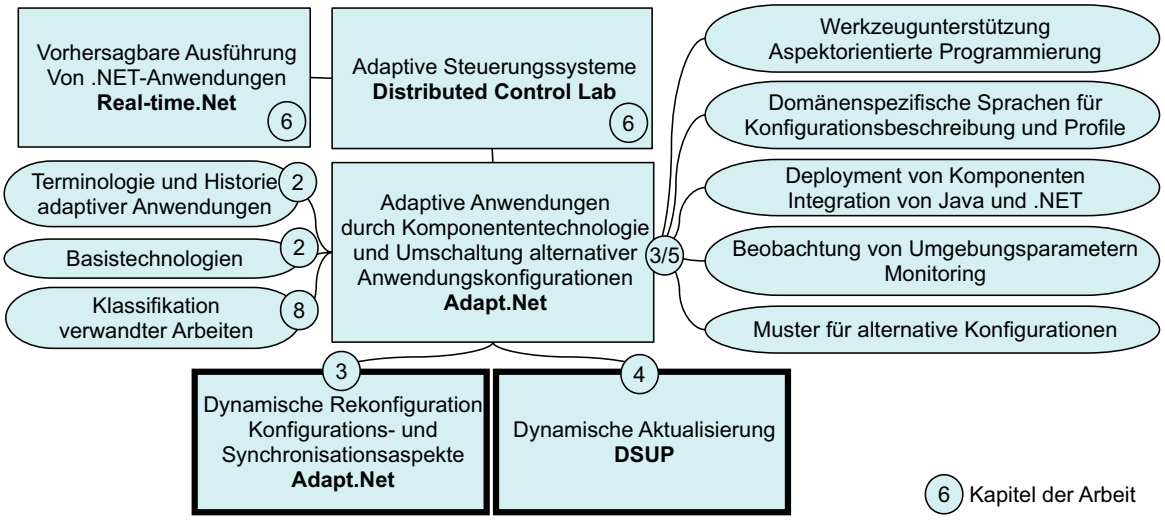

Abbildung 1.3: Strukur and Themengebiete der Arbeit