Bestimmung von Laufzeiteigenschaften mobiler Softwarearchitekturen

Volker Gruhn und Clemens Schäfer
Lehrstuhl für Angewandte Telematik / e-Business*, Institut für Informatik,
Fakultät für Mathematik und Informatik, Universität Leipzig
{gruhn,schaefer}@ebus.informatik.uni-leipzig.de

Abstract: Das Verhalten eines mobilen Systems wird durch seine Architektur (statische und dynamische Anteile, Softwareverteilung), die zu Grunde liegende Netzwerkinfrastruktur (Topologie, Parameter wie Bandbreiten oder Latenzzeiten) und Interaktionen der Benutzer mit dem System bestimmt. Um bereits zur Entwurfszeit ermitteln zu können, ob ein mobiles System nichtfunktionale Anforderungen an Antwortzeiten oder Verfügbarkeiten von Diensten erfüllt, kann eine Simulation des Systems auf Basis eines Architekturmodells unter Einbeziehung eines Netzwerk- und eines Benutzerinteraktionsmodells durchgeführt werden. Ein derartiger Ansatz unter Verwendung der Architekturbeschreibungssprache Con Moto wird in diesem Beitrag vorgestellt.

1 Motivation

Die Verwendung einer domänenspezifischen Architekturbeschreibungssprache (Architecture Description Language, ADL) zur Modellierung der Architektur mobiler verteilter Systeme wird als sinnvoll angesehen [GS05], denn der Einfluss der Mobilität verstärkt die Notwendigkeit, funktionale wie auch nichtfunktionale Eigenschaften einer Softwarearchitektur zu beschreiben und zu untersuchen. Dies korrespondiert mit der Aussage, dass Mobilität als das totale Abschmelzen aller Stabilitätsannahmen, die beim Distributed Computing üblicherweise getroffen werden, aufgefasst werden muss [RPM00] und deutet damit auf die Probleme hin, die bei der Entwicklung mobiler Systeme zu lösen sind. Zu den typischen Problemen zählen hierbei Netzwerkstrukturen, die nicht länger statisch sind und bei denen Kommunikationsknoten entstehen und verschwinden können, Kommunikationsfehler auf Grund von Verbindungsabbrüchen in Drahtlosnetzwerken oder eingeschränkte Konnektivität wegen (zu) niedriger Bandbreiten mobiler Kommunikationskanäle. Alle diese Probleme haben gemeinsam, dass sie die zur Laufzeit entstehenden nichtfunktionalen Eigenschaften des mobilen Systems wie Performanz, Robustheit oder Dienstgüte nachhaltig beeinflussen.

Mit unserer ADL *Con Moto* (italienisch für „mit Bewegung“) schlagen wir eine Sprache vor, die es Systementwicklern erlaubt, die genannten Aspekte bereits zur Entwurfszeit zu adressieren und adäquate Entwurfsentscheidungen für das mobile System zu treffen. In

*Der Lehrstuhl für Angewandte Telematik / e-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG.

diesem Papier gehen wir über die Grundzüge, die wir in [GS06] bereits ausgeführt haben, hinaus und vertiefen Aspekte wie die Simulation mobiler Softwarearchitekturen.

2 Einführung

Das Zusammenspiel von Mobilkommunikation und den damit verbundenen Unwägbarkeiten (variierende Netzabdeckung und Bandbreiten, Verbindungsabbrüche) mit Softwareverteilungsaspekten stellt die Entwickler von mobilen Systemen vor neue Herausforderungen. Einerseits sind mobile Systeme so zu entwerfen, dass sie bestimmte Dienstgütemerkmale möglichst gut erfüllen, andererseits soll die Umsetzung mit einem möglichst geringen technischen Aufwand erfolgen. Gerade diese Bewertung, nämlich welcher technische Aufwand getrieben werden muss, um bestimmte Dienstgüten sicherzustellen, ist im Moment nur schwer möglich.

Das Problem hierbei ist, dass grundsätzlich verschiedene Arten von Mobilität betrachtet werden müssen. Benutzer können mit ihren Geräten mobil sein (physikalische Mobilität), andererseits kann aber die eingesetzte Software auch ihren Ausführungsort wechseln (logische Mobilität). Bei der physikalischen Mobilität wird der Einfluss der Kommunikationsnetze deutlich, bei der logischen Mobilität müssen Aspekte wie zum Beispiel die unterschiedliche Leistungsfähigkeit von Hardwarekomponenten berücksichtigt werden.

Mit dem hier vorgeschlagenen Ansatz wollen wir eine Modellbildung ermöglichen, die im Gegensatz zu verwandten Arbeiten die Implikationen der Kommunikationsnetze und Hardwareaspekte auf die nichtfunkionalen Eigenschaften eines Softwaresystems berücksichtigt; zudem wollen wir dem Entwickler mobiler Systeme ein Werkzeug an die Hand geben, mit dem er bereits zur Entwurfszeit seine Designentscheidungen hinsichtlich ihrer Implikationen im späteren Betrieb überprüfen kann.

Daher sind wir weniger an Auswertungsergebnissen interessiert, die formale Methoden üblicherweise liefern, zum Beispiel dass ein System verklemmungsfrei ist. Von Interesse sind vielmehr Aussagen wie dass zum Beispiel eine Transaktion im System in 98% aller Fälle weniger als zwei Sekunden benötigt. Stochastische Methoden wie Markow-Ketten, die solchen Ergebnistypen erwarten lassen, sind jedoch hinsichtlich der Unterstützung von Mobilität unhandlich, so dass wir uns für einen Simulationsansatz entschieden haben.

Im Softwareentwicklungsprozess können die Simulationsergebnisse zur Bewertung eines initialen Architekturentwurfs verwendet werden. Dies kann in Form eines Szenarienvergleichs geschehen, bei dem unterschiedliche Ausprägungen des mobilen Systems hinsichtlich Benutzungsverhalten, Dimensionierung (Server, Netzwerkanteile) und Struktur untersucht werden. Aus den Simulationsergebnissen kann so die für ein mobiles System optimale Konfiguration hinsichtlich Struktur und Dimensionierung (wenn von einem vorgegebenem Benutzungsverhalten ausgegangen wird) ermittelt werden.

Die Herausforderung bei der Modellierung mobiler Systeme liegt hierbei in der Wahl eines angemessenen Abstraktionsniveaus, denn Übersimplifizierung würde zu bedeutungslosen Simulationsergebnissen führen, zu komplexe Modelle wären hingegen zur Entwurfszeit unpraktikabel. Zudem sollte der Modellierungsansatz so abstrakt und unabhängig wie

möglich von konkreten technischen Realisierungen mobiler Systeme sein, jedoch dürfen technologische Implikationen mobiler Systeme nicht völlig außer Acht gelassen werden, wenn aussagekräftige Simulationsergebnisse erzielt werden sollen.

3 Verwandte Arbeiten

ADLs stellen ein gut erforschtes Thema dar. Die Modellierung nichtfunktionaler Eigenschaften wurde bereits bei Shaw und Garlan [SG95] als relevant angesehen. Die Klassifikationsarbeiten von Medvidovic und Taylor [MT00] aus dem Jahr 2000 bieten einen breiten Überblick über die Eigenschaften verschiedener ADLs, wobei deutlich wird, dass die dort betrachteten ADLs keine Unterstützung für mobile Systeme bieten. Aus diesem Grund sind in letzter Zeit mobile ADLs entstanden [Oqu04, ITLS04], die die Dynamik mobiler Systeme abbilden können, was aus ihrem Ursprung im π-Kalkül herrührt; beide bieten jedoch keine Unterstützung nichtfunktionaler Eigenschaften. Neben den mobilen ADLs existieren weitere Arbeiten im Bereich nichtfunktionaler Eigenschaften. Sie basieren zum Teil auf Lamports TLA+ [Lam02], einer Logik zur Spezifikation nebenläufiger reaktiver Systeme. Zschaler [Zsc04] stellt eine Spezifikation temporaler Eigenschaften von komponentenbasierten Systemen vor, aber diese – ebenso wie die zu Grunde liegenden Arbeiten von Aagedal [Aag01] – lassen die Unterstützung von Mobilität vermissen. Andere Ansätze auf Basis von Markow-Ketten oder Prozessalgebren (z.B. die Arbeiten von Hermanns und Katoen [HK01]) greifen in diesem Aspekt ebenfalls zu kurz. Am verwandtesten mit unserem Ansatz sind sicher Arbeiten wie die von Bracchi und Cortellezza [BC04], die mobile Systeme auf Stochastic Activity Networks abbilden. Bei diesen Verfahren findet jedoch keine explizite Trennung zwischen physikalischen und logischen Konnektoren statt, was die Aussagekraft für die von uns angestrebten Auswertungsergebnisse reduziert. Dies trifft ebenfalls auf die Arbeit von Grassi, Mirandola und Sabetta [GMS04] zu, in der die Autoren ein Vorgehen zur Performanzanalyse auf Basis von UML-Modellen beschreiben. Zudem wird dort die Analyse sehr unhandlich, wenn man über die Zeit veränderliche Bandbreiten der Kommunikationsnetze zulässt. In der Arbeit von Reussner, Poernomo und Schmidt [RPS03] werden Zuverlässigkeitskennzahlen von Komponentendiensten aus den Zuverlässigkeitswerten von verwendeten Services und Markow-Modellen der betrachteten Services ermittelt. Dieses Verfahren findet jedoch bei einer dynamischen Rekomposition der Services zur Laufzeit seine Grenzen, also wenn Mobilität betrachtet werden soll.

4 Con Moto im Überblick

Abbildung 1 zeigt die verschiedenen Elemente von Con Moto. Die eigentliche Architekturbeschreibung (Architectural Description) besteht aus einer Spezifikation der Struktur (Structural Specification) und der Dynamik (Behavioral Specification) des Systems. Zusammen mit Instantiierungsinformationen (Instantiation Information) kann der Simulator das Architekturmodell instantiieren und auf Basis eines Modells des Kommunikati-

onsnetzwerks (Network Model) unter Berücksichtigung von Benutzerinteraktionsmustern (Usage Pattern) die gewünschten Eigenschaften der Architektur bestimmen.

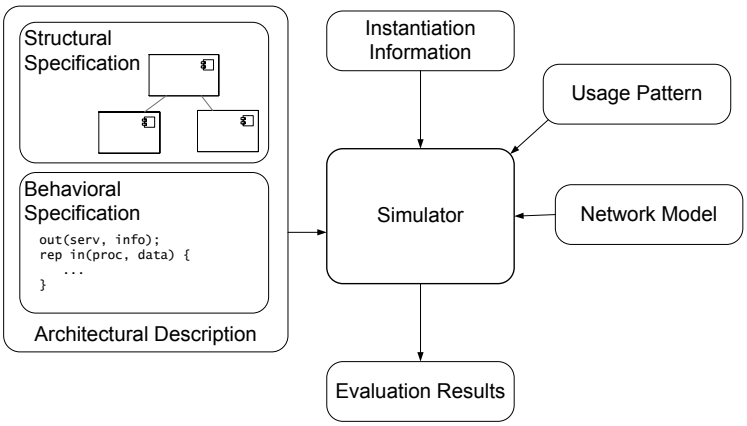

Abbildung 1: Bestandteile von Con Moto

Verhaltensmodell Wie andere mobile ADLs stützen wir das Verhaltensmodell von Con Moto auf Milners π-Kalkül [Mil99] ab. Beim π-Kalkül handelt es sich um eine Prozessalgebra mit expliziter Mobilitätsunterstützung, bei der Prozesse über so genannte Namen (die in Con Moto *Pins* heißen) kommunizieren. Analog zur Programmiersprache Pict [PT00], die ebenfalls auf dem π-Kalkül basiert, verwenden wir nur eine Untermenge des vollen π-Kalküls, was die (unbeabsichtigte) Modellierung von Nichtdeterminismen verhindert. Bei den Nachrichten, die zwischen den Prozessen kommuniziert werden, erlauben wir die Annotation einer simulierten Paketgröße, um die Transportzeiten von Nachrichtenpaketen über das Netzwerk abbilden zu können.

π-Kalkül	Con Moto	
$\overline{x}y$	`out(x,y)`	Synchrone Ausgabe
$x(y)$	`in(x,y)`	Eingabe
$e_1 \mid e_2$	`par e1, e2`	Parallele Komposition
$(\nu x)e$	`new x; e`	Kanalerzeugung
$!x(y).e$	`rep in(x,y) e`	Replizierende Eingabe

Tabelle 1: Notation

Wie in Tabelle 1 dargestellt, unterstützt Con Moto verschiedene Primitive, um Prozesse zu modellieren. Neben den Ein- und Ausgabeoperationen des π-Kalküls werden auch die Parallel- und Replikationsoperatoren unterstützt. Bei letzteren erfolgt jedoch eine Einschränkung auf Eingabeoperationen, was für das Erzeugen neuer Prozesse nach dem Empfang von Daten ausreichend ist.

Strukturelles Modell Wie bei ADLs allgemein üblich, besteht das strukturelle Modell auch bei Con Moto aus Komponenten, Konnektoren und Konfigurationen.

Die Komponenten stellen den Ort dar, an dem die eigentlichen Berechnungen stattfinden. Bei Con Moto unterscheiden wir explizit zwischen *physikalischen* und *logischen* Komponenten. Physikalische Komponenten bilden Geräte ab, also Hardware mit Ressourcenbeschränkungen, während logische Komponenten die eigentlichen Softwarekomponenten repräsentieren, die sowohl als Komponenten (zustandslos) als auch als Komponenteninstanzen (zustandsbehaftet) vorkommen können.

Konnektoren bilden die Interaktionen zwischen Komponenten ab. In Con Moto unterscheiden wir dabei explizit zwischen *physikalischen* und *logischen* Konnektoren. Logische Konnektoren sind ideal und stellen die Kommunikationsbeziehungen zwischen logischen Komponenten dar, wogegen physikalische Konnektoren über eine beschränkte Bandbreite und Latenzzeiten größer Null verfügen und die Kommunikation zwischen physikalischen Komponenten realisieren. Logische Konnektoren sind immer in physikalische Konnektoren eingebettet.

Durch diese Art der Modellierung und die Möglichkeit, logische Komponenten wie auch logische Konnektoren zwischen Prozessen zu kommunizieren, können die in [MPR99] genannten Arten von Mobilität, also Client-server, Remote evaluation, Code-on-demand und Mobile agents, realisiert werden.

5 Grundsätzliches Vorgehen bei der Simulation

Gemäß der Klassifikation in [Jai91] greifen wir bei unserer Simulation mobiler Softwarearchitekturen auf zeitkontinuierliche zustandsdiskrete Modelle zurück. Die Darstellung eines Systems während der Simulation erfolgt wie in Abbildung 2 dargestellt in drei Schichten (die dargestellten Komponenten entsprechen dem Beispiel aus Abbildung 3). Aus der Architekturbeschreibung wird die so genannte *Definitionsschicht* erzeugt. Diese stellt alle architekturellen Informationen dar und kann im Simulator instantiiert werden. Somit enthält die *Instanzschicht* die logische Architektur zur Laufzeit des Systems. Hinzu kommt in der Simulation eine *Kommunikationsschicht*, in der der eigentliche Datenaustausch zwischen den instantiierten Komponenten einer Architektur abgewickelt wird.

Definitionsschicht Beim Laden einer Architekturbeschreibung wird in der Definitionsschicht zunächst das initiale Architekturmodell erzeugt. Dieses enthält die Struktur des Systems (Komponenten) sowie die Beschreibung der dynamischen Anteile (Prozesse auf den Komponenten). Aus den Beschreibungen des Netzwerkmodells wird in der Kommunikationsschicht auch die initiale Netzwerkstruktur erzeugt, ebenso die global verfügbare Infrastrukturkomponente ‚Netzwerkmanager'. Diese sorgt für Infrastrukturdienste wie das Routing im Netzwerk oder das Auffinden von Komponenten.

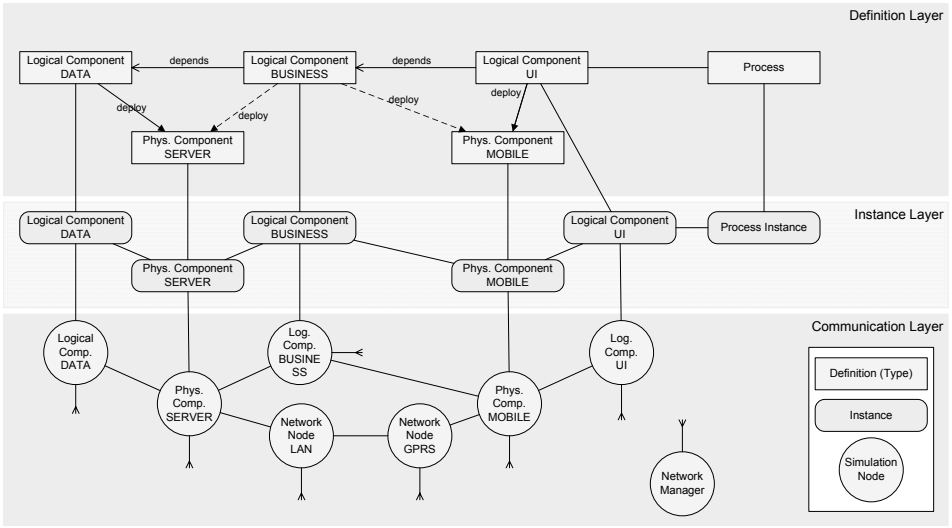

Abbildung 2: Schichten während der Simulation

Instanzschicht Die Instanzschicht bildet den logischen Zustand des Systems zur Laufzeit ab. Daher werden für alle Komponenten in dieser Schicht Komponenteninstanzen angelegt. Nur diese Instanzen können aktiv an der Simulation teilnehmen. Aufgrund der Beschreibung im Instantiiation Model wird zu Beginn einer Simulation eine Menge von physikalischen Komponenteninstanzen angelegt. Auf diesen physikalischen Komponenteninstanzen, die Hardware-Komponenten entsprechen, werden dann logischer Komponenteninstanzen angelegt, wie dies die initiale Deployment-Beschreibung in der Architektur vorgibt. Eventuell vorhandene Initialisierungsprozesse auf physikalischen Komponenteninstanzen beziehungsweise auf logischen Komponenteninstanzen werden quasi als Bootstrap-Prozesse instantiiert.

Kommunikationsschicht In der Kommunikationsschicht sind Knoten enthalten, die wie in [Tos05] beschrieben einem Ansatz verallgemeinerter zellularer Automaten folgen. In jedem Simulationsschritt übernehmen diese Knoten die Daten an ihren Eingängen und ermitteln aus diesen Daten sowie ihrem inneren Zustand einen Folgezustand, wobei sie gegebenenfalls Ausgangsdaten erzeugen. Im Gegensatz zu herkömmlichen zellularen Automaten, bei denen die Kommunikationsbeziehungen zwischen einzelnen Knoten durch die räumliche Anordnung in einem Grid definiert sind, sind in unserem System Verbindungen zwischen beliebigen Knoten möglich, die zudem zur Laufzeit variabel sind.

Zur Simulation wird nun für jede physikalische Komponenteninstanz ein Simulationsknoten gebildet, der wiederum mit Netzwerkknoten verbunden sein kann. Logische Komponenteninstanzen werden ebenfalls auf Simulationsknoten abgebildet, die mit Knoten der physikalischen Komponenteninstanzen verbunden sind. Prozessinstanzen werden innerhalb der logischen (und physikalischen) Komponenteninstanzen gehalten und nicht auf

eigenständige Simulationsknoten abgebildet, was jedoch eine Implementierungsentscheidung ist.

Dynamik und abstrakte Middleware Das Verhalten des mobilen Systems ist durch die auf dem π-Kalkül basierende Sprache gegeben. Diese unterstützt jedoch nur Primitive zur Kommunikation. Um eine realistische Abbildung (und vor allem effiziente Modellierung) mobiler Aspekte zu erreichen, stellt Con Moto eine Reihe von Befehlen zur Verfügung, die quasi eine abstrakte Middleware darstellen. Dies sind unter anderem

- Lookup-Befehle, die Referenzen auf physikalische Komponenteninstanzen und logische Komponenten liefern,
- Connect-Befehle, die das Verbinden von Ports (unter Verwendung der Referenzen) erlauben, indem sie implizite Konnektoren erzeugen und diese durch das Verbinden von Pins, also Namen in den π-Kalkül-Prozessen, in das Prozessmodell einbringen,
- Instantiate-Befehle, die das Instantiieren von Komponenten erlauben und
- Deploy- / Undeploy-Befehle, die das Verschieben von logischen Komponenten / Komponenteninstanzen ermöglichen.

Diese Befehle können in Con Moto wie Methoden verwendet werden, die auf Objekten (in Con Moto: Referenzen auf Komponenten) aufgerufen werden. Beispielsweise kann mit

```
aLogRef = aPhysRef.lookup('ALogCompName');
```

der Variable `aLogRef` eine Referenz auf eine logische Komponente zugewiesen werden, die unter dem Namen `ALogCompName` auf der physikalischen Komponenteninstanz `aPhysRef` verteilt ist. Mittels

```
connect(aLogRef.PORT1, anotherLogRef.PORTn);
```

ließe sich der Port `PORT1` auf der logischen Komponenteninstanz `aLogRef` mit dem Port `PORTn` der logischen Komponenteninstanz `anotherLogRef` verbinden und die dazu nötigen impliziten Konnektoren erzeugen.

Mobiler Code Da in Con Moto logische Komponenten und logische Komponenteninstanzen gewöhnliche Kommunikationsobjekte darstellen, kann die Mobilisierung von Code durch das Lösen der aktuellen Bindung einer logischen Komponenteninstanz an eine physikalische Komponenteninstanz realisiert werden, falls danach ein Transfer der serialisierten Informationen der logischen Komponente (oder Komponenteninstanz) an die physikalische Ziel-Komponenteninstanz stattfindet und eine Bindung der deserialisierten logischen Komponenteninstanz an die physikalische Komponenteninstanz erfolgt.

6 Anwendungsbeispiel

Im Folgenden präsentieren wir ein einfaches mobiles Client/Server-System. Die Benutzer des Systems verfügen über mobile Endgeräte, die über jeweils einen Mobilfunkkanal mit einem Server verbunden sind; hierfür sehen wir entweder eine GPRS-Verbindung mit relativ niedriger Bandbreite oder eine UMTS-Verbindung mit entsprechend höherer Bandbreite vor. In unserem Beispielsystem existieren drei Softwarekomponenten: die Benutzeroberfläche (UI) wird auf den mobilen Endgeräten verteilt. Die Persistenzkomponente (DATA) ist nur auf dem Server verfügbar, wogegen die Komponente mit der Geschäftslogik (BUSINESS) entweder auf dem Server oder aber auf den mobilen Endgeräten verteilt sein kann. Ruft der Benutzer über die Benutzeroberfläche einen Dienst auf, sendet die Komponente UI eine Dienstanfrage an die Komponente BUSINESS, die ihrerseits wiederum einen Service von DATA aufruft. Wenn dieser ein Ergebnis zurückliefert, gibt BUSINESS dieses an UI weiter, wo das Ergebnis dem Benutzer angezeigt wird. Die Struktur dieses Systems wird in Abbildung 3 verdeutlicht.

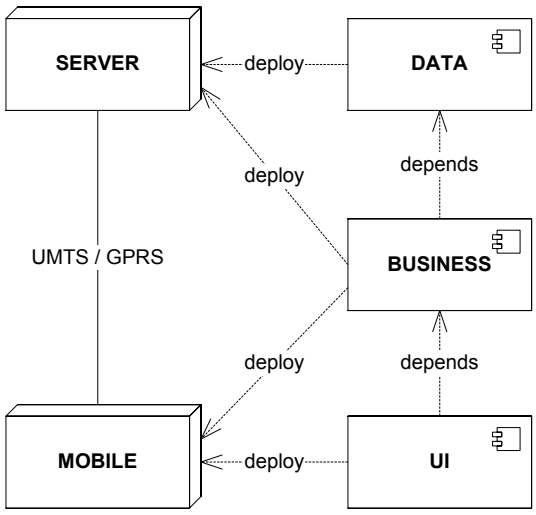

Abbildung 3: Beispielsystem

6.1 Modellierung in Con Moto

In der Con Moto-Beschreibung dieses Systems (einem XML-Dokument) werden zwei physikalische Komponenten `MOBILE` und `SERVER` deklariert, die jeweils mit Rechenleistung und möglichen Netzwerkverbindungen, die während der Simulation zu physikalischen Konnektoren führen, parametrisiert werden. `MOBILE` Komponenten können sich daher entweder mit Netzwerkknoten vom Typ UMTS oder GPRS verbinden, während für die `SERVER` Komponente nur eine Verbindung mit einem WAN möglich ist.

Im Netzwerkmodell werden die Netzwerktypen UMTS, GPRS und WAN definiert und die Bandbreiten (10.0, 2.0 und 1000.0 kBit/s) gesetzt. Ein zusätzlicher Netzwerkknoten namens `backbone` dient zur Verknüpfung aller Kommunikationsinstanzen und erlaubt so die direkte Adressierung aller physikalischer Komponenten, wie dies im Internet durch IP-Adressen auch der Fall ist.

Durch die Einführung von Ports und Porthierarchien werden Schnittstellen zum Veröffentlichen und Aufrufen von Methoden definiert. Entsprechende Makros in diesen Schnittstellen realisieren die Kommunikationsprotokolle für die Dienstaufrufe im π-Kalkül. Durch diese Makros und deren Wiederverwendung in Verbindung mit den Porthierarchien ist es möglich, in Con Moto das eigentliche System strukturell äquivalent zu einer imperativen Programmbeschreibung zu definieren.

Für die logischen Komponenten `DATA`, `BUSINESS` und `UI`, die ebenfalls Bestandteil des Con Moto-Modells sind, werden Start-Up Prozesse definiert, die bei Instantiierung der Komponenten ausgeführt werden und die entsprechenden Lookup- und Konnektivitätsfunktionen durchführen. Die eigentlichen Services auf den logischen Komponenten werden ebenfalls in Prozessen im π-Kalkül ausgedrückt und sind im Beispiel hier durch das Blockieren der CPU für eine bestimmte Zeit (100 ms bei `DATA` und 500 ms bei `BUSINESS`) und das Zurückliefern von Datenblöcken definierter Größe (100 Byte bei `DATA` und 5000 Byte bei `BUSINESS`) modelliert.

6.2 Simulation

Das hier beschriebene Beispielsystem wurde mit Hilfe des prototypisch implementierten Con Moto-Simulators einer Simulation mit 10 bis 120 Benutzern, also mit 10 bis 120 MOBILE Geräten, unterzogen. Die Benutzerinteraktionen mit dem System wurden durch einen Poisson-Prozess mit einer Ankunftsrate von 10 Ereignissen pro Stunde modelliert. Die gesamte Simulation wurde mit einer Zeitauflösung von einer Millisekunde durchgeführt.

Unterschieden wurden bei dem System drei Szenarien.

- Zunächst wurde von einem Deployment der Komponente BUSINESS immer auf dem Server ausgegangen (1. Szenario, ‚-mob‘). Hierbei zeigt sich in den Ergebnissen in Abbildung 4, dass ab 90 Benutzern das System für die Kanäle GPRS und UMTS zunehmend überlastet ist und die durchschnittlichen Antwortzeiten der Dienste erheblich ansteigen.
- Verteilt man nun beim System die Komponente BUSINESS ex ante auf die Komponente MOBILE (2. Szenario, ‚+mob -deploy‘), so ist im Fall GPRS die Antwortzeiten auch in dem Bereich höherer Nutzerzahlen konstant. Für UMTS verhält sich das System analog, die Werte wurden jedoch aus Gründen der Übersichtlichkeit weggelassen.
- Wird die Komponente BUSINESS zur Laufzeit des Systems über den mobilen Kommunikationskanal auf MOBILE verteilt (3. Szenario, ‚+mob +deploy‘), so steigt die

durchschnittliche Antwortzeit unter UMTS an, bleibt aber ebenfalls über die Laufzeit des Systems im Rahmen der betrachteten Nutzerzahlen konstant. Hierbei wird der deutliche Anteil der Komponentenübertragung an der durschnittlichen Antwortzeit deutlich.

Zur Messung der Antwortzeiten wurde im Simulationsmodell die Zeitdifferenz zwischen dem Absetzen eines Aufrufs in der Komponente UI und dem Eintreffen des Rückgabewerts gemessen. Weitere Möglichkeiten zu Messung wäre die Auslastung von Netzwerkknoten, was an dieser Stelle jedoch nicht erfolgt ist.

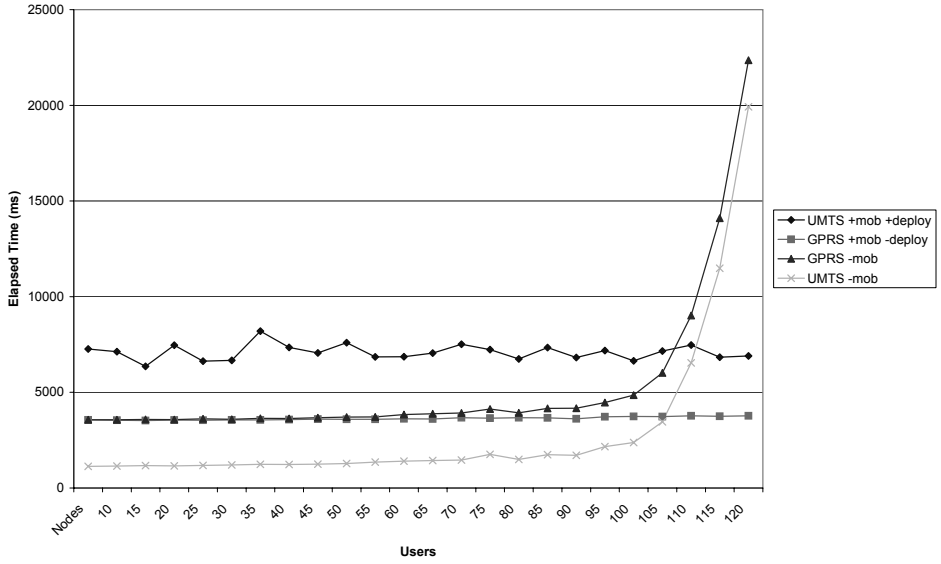

Abbildung 4: Simulationsergebnisse

7 Diskussion und Ausblick

In diesem Beitrag haben wir beschrieben, wie mobile Systeme mit Con Moto mit dem Ziel modelliert werden können, zur Entwurfszeit Dienstgüteparameter des Systems durch Simulation zu bestimmen. Indem wir eine Architekturbeschreibung auf Basis des π-Kalküls vorgenommen und dabei eine klare Unterscheidung zwischen physikalischen und logischen Komponenten und Konnektoren vorgenommen haben, ist die Modellierung der mobilen Systeme auf einem recht hohen Abstraktionsniveau mit vertretbarem Aufwand möglich. Erste Simulationsversuche für ein einfaches Beispiel zeigen, dass der Ansatz allgemein Erfolg versprechend ist.

Weitere Arbeiten umfassen die Untersuchung von Modellen für physikalische Kommunikationskanäle, die bislang lediglich durch die Annahme einer konstanten Bandbreite

und Latenzzeit modelliert werden. Hierbei sind komplexere Modelle des Übertragungsverhaltens denkbar und für realistische Simulationsergebnisse auch nötig, die schwankende Bandbreiten, unterschiedliche Latenzzeiten oder sogar Verbindungsabbrüche zulassen. Ebenso im Bereich der Modellierung von Benutzerinteraktionen mit dem System ergibt sich weiterer Forschungsbedarf, wobei jedoch nicht nur stochastische Prozesse tiefere Betrachtung erfahren sollten, sondern auch Aspekte, wie die Interaktionsmuster für die Simulation aus gegebenenfalls vorhandenen Geschäftsprozessmodellen abgeleitet werden können. Eine weitere zukünftige Aufgabe ist schließlich die Evaluierung des vorgestellten Ansatzes durch Vergleich von Simulationsergebnissen mit belastbaren Messergebnissen aus tatsächlich implementierten Systemen. Diese Validation wird letztlich erlauben, Aussagen über die Anwendbarkeit und Nützlichkeit des vorgestellten Ansatzes zu treffen.

Literatur

[Aag01] Jan Øyvind Aagedal. *Quality of Service Support in Development of Distributed Systems*. Dissertation, University of Oslo, 2001.

[BC04] Paola Bracchi und Vittorio Cortellessa. A framework to model and analyze the performability of mobile software systems. In *WOSP '04: Proceedings of the 4th international workshop on Software and performance*, Seiten 243–248, New York, NY, USA, 2004. ACM Press.

[GMS04] Vincenzo Grassi, Raffaela Mirandola und Antonino Sabetta. UML based modeling and performance analysis of mobile systems. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, Seiten 95–104, New York, NY, USA, 2004. ACM Press.

[GS05] Volker Gruhn und Clemens Schäfer. Architecture Description for Mobile Distributed Systems. In *Proceedings of the Second European Workshop on Software Architecture (EWSA 2005)*, Seiten 239–246. Springer-Verlag Berlin Heidelberg, 2005.

[GS06] Volker Gruhn und Clemens Schäfer. Modellierung und Analyse mobiler Architekturen. In *INFORMATIK 2006. Beiträge der 36. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, 2006.

[HK01] Holger Hermanns und Joost-Pieter Katoen. Performance Evaluation := (Process Algebra + Model Checking) $\times$ Markov Chains. In *Proceedings of CONCUR 2001*, LNCS 2154, Seiten 59–81. Springer-Verlag Berlin Heidelberg, 2001.

[ITLS04] Valérie Issarny, Ferda Tartanoglu, Jinshan Liu und Françoise Sailhan. Software Architecture for Mobile Distributed Computing. In *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, Seiten 201–210. IEEE, 2004.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[MPR99] Cecilia Mascolo, Gian Pietro Picco und Gruia-Catalin Roman. A fine-grained model for code mobility. In *ESEC/FSE-7: Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*, Seiten 39–56, London, UK, 1999. Springer-Verlag.

[MT00] Nenad Medvidovic und Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[Oqu04] Flavio Oquendo. π-ADL: An Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, 29, May 2004.

[PT00] Benjamin C. Pierce und David N. Turner. Pict: A Programming Language Based on the Pi-Calculus. In G. Plotkin, C. Stirling und M. Tofte, Hrsg., *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[RPM00] Gruia-Catalin Roman, Gian Pietro Picco und Amy L. Murphy. Software Engineering for Mobility: A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, Seiten 241–258. ACM Press, 2000.

[RPS03] Ralf H. Reussner, Iman H. Poernomo und Heinz W. Schmidt. Contracts and Quality Attributes of Software Components. In *Proceedings of the Eighth International Workshop on Component-Oriented Programming (WCOP)*, 2003.

[SG95] Mary Shaw und David Garlan. Formulations and Formalisms in Software Architecture. In Jan van Leeuwen, Hrsg., *Computer Science Today: Recent Trends and Developments*, Jgg. 1000 of *Lecture Notes in Computer Science*, Seiten 307–323. Springer, 1995.

[Tos05] Predrag Tosic. Cellular Automata for Distributed Computing: Models of Agent Interaction and Their Implications. In *IEEE Int'l Conf. on Systems, Man and Cybernetics*, 2005.

[Zsc04] Steffen Zschaler. Formal Specification of Non-functional Properties of Component-Based Software. In Jean-Michel Bruel, Geri Georg, Heinrich Hussmann, Ileana Ober, Christoph Pohl, Jon Whittle und Steffen Zschaler, Hrsg., *Workshop on Models for Non-functional Aspects of Component-Based Software (NfC'04) at UML conference 2004*, September 2004.