Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Diplomarbeit

Management mobiler Prozesse:

Logging, Monitoring und Recovery

Juli 2009

Benjamin Böhling

3boehlin@informatik.uni-hamburg.de
Studiengang Informatik
Matr.-Nr. 5609538
Fachsemester 11

Erstgutachter: Professor Dr. Lamersdorf
Zweitgutachter: Professor Dr. Dreschler-Fischer

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird zunächst das Thema der Diplomarbeit motiviert. Im Anschluss wird die genaue Zielsetzung aufgezeigt und die Vorgehensweise erläutert.

1.1 Motivation

Zu Beginn des Computerzeitalters waren Computer große und verglichen mit ihren heutigen „Verwandten“ auch sehr kostenaufwändige Geräte, die als so genannte Mainframes in Firmen mit ausreichenden finanziellen Mitteln zum Einsatz kamen. Der dominierende Ansatz zu dieser Zeit war der zentrale Ansatz, d.h., die einzelnen Großrechner waren zumeist gar nicht miteinander vernetzt, sondern ermöglichten lediglich den Zugriff mittels so genannter „dummer Terminals“, die im Grunde nichts weiter als ein Bildschirm mit Tastatur waren. Durch die Entwicklung von immer kleineren und leistungsfähigeren Komponenten und dem rapiden Fall der Kosten für Rechner und ihrer Bauteile kam es in der Folgezeit zur Verbreitung von Personalcomputern, die auch für Privatpersonen erschwinglich waren bzw. sind. Anstelle eines teuren Großrechners konnten nun viele kleine kostengünstige Rechner verwendet und dank der ebenfalls stetigen Verbesserung der Netzwerktechnologie miteinander verbunden werden, um gemeinsam Probleme und Aufgaben zu lösen bzw. zu kooperieren. Die monolithischen „Informationsinseln“ wurden aufgrund dieser Entwicklung und ihrer Vorteile größtenteils von verteilten Systemen abgelöst.

Heute ist der genannte Entwicklungsprozess so weit fortgeschritten, dass eine Vielzahl an mobilen Geräten existiert (z. B. Mobiltelefon, Notebook, PDA), die es dem Nutzer prinzipiell erlaubt, ortsungebunden mit anderen in der Umgebung befindlichen Geräten und kommunikationstauglichen („smarten“) Objekten, ggf. unter Berücksichtigung von Kontextinformationen, zu interagieren. Eine Vernetzung mobiler Geräte zur Umsetzung von mobilen Systemen birgt hierbei Möglichkeiten wie die Nutzung ortsspezifischer Dienste (z. B. das Auffinden einer Tankstelle in der Nähe) und könnte darüber hinaus in naher Zukunft eingesetzt werden, um Katastrophen wie z. B. Waldbrände auf Basis kleiner, drahtlos miteinander vernetzter Sensoren zu entdecken bzw. sogar zu verhindern.

Damit eine nahtlose Kooperation zwischen den mobilen Geräten ermöglicht werden kann, müssen allerdings die speziellen Randbedingungen berücksichtigt werden, denen die mobilen Geräte im Gegensatz zu ihren fest verkabelten, permanent an das Stromnetz angeschlossenen und zumeist leistungsfähigeren Pendants unterliegen. Vor allem die hohe Fehleranfälligkeit und geringere Leistungsfähigkeit, die aufgrund der Portabilität bei diesen Endgeräten inhärent sind sowie die sich ständig aufgrund der Mobilität verändernde Umgebung und Topologie des Rechnernetzes, verbunden mit der volatilen

und bezüglich der Leistungsparameter stark schwankenden drahtlosen Kommunikation führen dazu, dass angepasste Maßnahmen getroffen werden müssen, um die Zuverlässigkeit auch in so einer in vielerlei Hinsicht unbeständigen Umgebung zu gewährleisten. Diese Maßnahmen zur Verbesserung der Fehlertoleranz sollten hierbei möglichst automatisch, also ohne manuelle Einwirkung der jeweiligen Benutzer, ausgeführt werden, um die erhöhte Komplexität des Gesamtsystems, die mit derartigen Maßnahmen einhergeht, vom Benutzer fernzuhalten und um die Handhabbarkeit aus Sicht der Anwender nicht zu verschlechtern.

Um einen höheren Gütegrad in Bezug auf die Kooperation zu erreichen und damit auch komplexere Aufgaben adäquat und effizient zu bearbeiten, bietet sich das Konzept der *mobilen Prozesse* an. Im Kern versteht man hierunter Prozesse, die zwischen Geräten inklusive der aktuellen Statusinformationen migrieren können, um dadurch die Beschränkungen und Möglichkeiten einzelner Geräte zu umgehen und das Potenzial leistungsfähigerer Geräte in der näheren Umgebung auszuschöpfen. [KZL07a] Da bei einer derartigen Migration die Kontrolle über den „eigenen“ Prozess bzw. die Prozessausführung abgegeben wird, ist darüber hinaus eine erweiterte Managementfunktionalität wünschenswert, um den Verarbeitungsprozess im Rechnernetz nachvollziehen und kontrollieren zu können.

Die folgende Arbeit beschäftigt sich daher unter Berücksichtigung der Herausforderungen, die aus den speziellen Restriktionen bei der Ad-hoc-Kooperation zwischen mobilen Geräten resultieren, mit einer Managementinfrastruktur zur Unterstützung und Verbesserung der Qualität derartiger volatiler Systeme auf Basis mobiler Prozesse.

1.2 Zielsetzung

Während in konventionellen stationären verteilten Systemen Ausfälle und Verbindungsabbrüche von bzw. zwischen Knoten eher zur Ausnahme gehören, sind derartige Fehler in mobilen Systemen aufgrund der Verwendung von drahtloser Kommunikation, fehleranfälligen mobilen Endgeräten und allgemein der Mobilität der Teilnehmer häufig die Regel. Diese hohe Fehleranfälligkeit muss bei der Entwicklung von Infrastrukturen für mobile Systeme berücksichtigt werden, weshalb eine speziell auf die Eigenschaften dieser Systeme abgestimmte Managementfunktionalität vonnöten ist.

Ziele dieser Diplomarbeit sind daher die Erarbeitung und Entwicklung von Konzepten, die das Management von mobilen Prozessen ermöglichen sowie die prototypische Implementierung einer solchen Managementumgebung. Unter dem weitläufigen Begriff Management sind in diesem Kontext Maßnahmen zur Unterstützung und Sicherung des Ablaufes von mobilen Prozessen gemeint, also eine adäquate und automatisierte Fehlererkennung und Fehlerbehandlung. Neben Recovery-Maßnahmen werden Logging- und Monitoring-Mechanismen angestrebt, die es ermöglichen, die Ausführung und den durch die Migration beschriebenen Ausführungspfad dieser speziellen Prozesse nach-

vollziehen zu können. In diesem Zusammenhang ist beispielsweise relevant, welche Knoten in welcher Reihenfolge an der Prozessbearbeitung beteiligt waren oder auf welchem Knoten sich derzeit ein laufender Prozess befindet. Des Weiteren gehören Maßnahmen zur Sicherstellung der Rückgabe möglicher, während der Prozessausführung produzierter Zwischen- und Endergebnisse zum Prozessinitiator dazu.

1.3 Vorgehensweise

Im Anschluss an dieses einleitende Kapitel werden in Kapitel zwei die Grundlagen des Mobile Computing verdeutlicht, um zunächst einen generellen Überblick über das Gebiet und seine Fassetten zu gewährleisten. Das Kapitel setzt sich aus drei Teilkapiteln zusammen: Der erste Teil behandelt verteilte Systeme, um mithilfe der dort gewonnenen Erkenntnisse im zweiten Teil eine Spezialform dieser Kategorie, die mobilen Systeme und ihre Besonderheiten, zu beschreiben. Der dritte Teil beschreibt „mobile Prozesse“, die ein wichtiges Kooperationskonzept in mobilen Systemen sind.

Das darauf folgende Kapitel behandelt den Aspekt des Managements mobiler Prozesse. Neben der Definition und Abgrenzung des weitläufigen Begriffs werden hier das Management bzw. relevante Aspekte des Managements in stationären Systemen behandelt, um auf Basis dieser Erkenntnisse bei Berücksichtigung der Unterschiede zu mobilen Systemen angepasste Konzepte zu entwickeln und Anforderungen abzuleiten, die es ermöglichen, solche Funktionalitäten in einer mobilen Umgebung einzubetten. Darüber hinaus werden hier mögliche Umsetzungsarten der Managementinfrastruktur diskutiert, verglichen und bewertet.

Das vierte Kapitel beschreibt die Entwicklung einer Managementinfrastruktur für mobile Systeme unter Berücksichtigung der in den vorherigen Kapiteln gesammelten Erkenntnisse. Hierbei wurden wenn möglich bereits etablierte Konzepte in, an das Management mobiler Prozesse angepasster Form, verwendet und wenn notwendig eigene Konzepte entwickelt.

Kapitel fünf befasst sich mit der Implementierung der Funktionalität in die im Forschungsprojekt *Distributed Environment for Mobility-Aware Computing* (DEMAC) entwickelte Middleware für mobile Prozesse, um die entwickelten Konzepte und Ideen zu validieren. Bei der Entwicklung wurden die Restriktionen von mobilen Endgeräten berücksichtigt, indem darauf geachtet wurde, dass die Managementkomponente möglichst ressourcensparend einsetzbar ist und sich zudem aufwandsarm erweitern lässt.

Das sechste und gleichzeitig letzte Kapitel dieser Diplomarbeit beinhaltet eine Zusammenfassung der gewonnenen Resultate im Rahmen eines Fazits und gibt einen Ausblick.

2 Grundlagen des Mobile Computings

2.1 Verteilte Systeme

Die Einführung in die Thematik beschäftigt sich zunächst allgemein mit verteilten Systemen, um eine Charakterisierung von mobilen Systemen als besondere Ausprägung verteilter Systeme zu erleichtern. Zentrale Aspekte sind hierbei neben der eigentlichen Definition Anforderungen an verteilte Systeme sowie der Transparenzbegriff bzw. die Transparenzarten. Vor allem Letzteres spielt in Bezug auf die geplante Managementfunktionalität eine wichtige Rolle, da sämtliche Maßnahmen zur Verbesserung der Zuverlässigkeit innerhalb des jeweiligen Netzwerkes möglichst automatisch von der Middleware durchgeführt und damit verborgen, d.h., transparent für die einzelnen Teilnehmer gehalten werden sollten.

2.1.1 Definition und Beispiel eines verteilten Systems

In der Literatur existieren diverse zum Teil sehr unterschiedliche Definitionen für verteilte Systeme. *Tannenbaum* charakterisiert ein verteiltes System wie folgt:

> „*Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen.*“ [TS08, S. 19]

Nach Tannenbaum wird ein verteiltes System folglich dadurch charakterisiert, dass es sich aus autonomen miteinander vernetzten Komponenten zusammensetzt und die Aspekte der Verteilung für die Benutzer oder auch Anwendungen transparent sind, d.h. verborgen bleiben. Implizit wird darüber hinaus Kooperation gefordert, da sonst die Bedingung, dass die einzelnen Komponenten wie ein einziges System erscheinen, nicht erfüllt werden kann. Die Art der Computer wird völlig offen gelassen, d.h., sowohl leistungsstarke Mainframes als auch kleine Knoten in Sensornetzen sind prinzipiell denkbar. Darüber hinaus werden keine Annahmen über die Art der Vernetzung getroffen. [TS08]

Der derzeit größte und wohl bekannteste Vertreter eines verteilten Systems ist das Internet. Die Nutzung des Internets ermöglicht Benutzern nahezu von überall den Zugriff auf Services wie das World Wide Web, E-Mail oder Dateitransfer. [CDK05] Es setzt sich aus Millionen von heterogenen, vernetzen Rechnern zusammen die über eine einheitliche Sprache (TCP-IP Protokollstack) miteinander kommunizieren und kooperieren. Ein Protokoll definiert hierbei die Sprache und Regeln der Kommunikation zwischen den einzelnen Knoten. [Ham05]

2.1.2 Gründe für die Verteilung

Nur weil der technische Fortschritt in Bezug auf die Netzwerktechnologie und die Miniaturisierung mittlerweile verteilte Anwendungen ermöglicht, stellt sich vor allem angesichts der Tatsache, dass verteilte Anwendungen häufig sehr zeit- und kostenaufwändig sind, die Frage, welche Gründe für den Einsatz eines verteilten anstelle eines zentralen Ansatzes sprechen [Ham05]:

- **Gemeinsame Nutzung von Hardware:** Durch die Verwendung eines verteilten Systems können Hardwareressourcen wie Drucker, Scanner oder andere teure Peripheriegeräte gemeinsam genutzt werden, um Kosten zu sparen. [Ham05]
- **Parallelisierung:** In verteilten Systemen ist es möglich, die Verarbeitungsleistung durch parallele Ausführung der Aufgaben zu verbessern. [SS07]
- **Lastausgleich/Ausfallsicherheit:** Indem Daten redundant auf unterschiedlichen Maschinen gespeichert werden oder verschiedene Server redundante Funktionalität anbieten, können Ausfälle einzelner Server ausgeglichen und ggf. vollständig kompensiert werden. Gleichzeitig ermöglicht diese Redundanz die Verteilung der Last vieler Benutzer auf die einzelnen Server, womit Engpässe vermieden werden können. [SS07]
- **Inkrementelle Erweiterbarkeit:** Reicht die Leistung eines Systems aufgrund gestiegener Anforderungen nicht mehr aus, ist eine Erweiterung vonnöten. Während im zentralen Ansatz im schlimmsten Fall der gesamte Mainframe durch einen leistungsfähigeren ersetzt werden muss, lässt sich ein verteiltes System schrittweise flexibel erweitern, z. B. durch das Hinzufügen eines weiteren Servers. [Ben04]
- **Wirtschaftlichkeit:** Im Gegensatz zu Großrechnern haben Mikroprozessoren ein besseres Preis-/Leistungsverhältnis. [Web98]

2.1.3 Ziele von verteilten Systemen

Im Folgenden werden einige grundlegende Anforderungen bzw. Ziele von verteilten Systemen vorgestellt.

Ressourcenzugriff

Eines der wichtigsten Ziele eines verteilten Systems ist es, den einzelnen Benutzern und Anwendungen den Zugriff auf entfernte, nicht private Ressourcen zu ermöglichen und zu erleichtern, sodass diese effizient und kontrolliert gemeinsam genutzt werden können. Beispiele für Ressourcen sind Peripheriegeräte wie Drucker, Scanner, aber auch Festplatten, Webseiten, Netzwerke, Dateien und Datenbanken. Die Umsetzung dieser Anforderung erleichtert die Kooperation und den Informationsaustausch und hat darüber hinaus wirtschaftliche Vorteile (vgl. 2.1.2 Punkt 1). [TS08]

Verteilungstransparenz

> „*Ein verteiltes System, das in der Lage ist, sich Benutzern und Anwendungen so darzustellen, als sei es nur ein einziges Computersystem, wird als transparent bezeichnet.*“ [TS08, S. 21]

Um eine einheitliche Sicht auf das System zu erlangen und damit die Tatsache der Verteilung zu verbergen, wird demnach das Prinzip des *Information Hiding* angewandt. Unter *Information Hiding* wird in diesem Kontext das „Verstecken“ von Informationen wie z. B. technischer Details oder allgemein die Abstraktion von Komplexität verstanden. [Kel99] Das *Open Distributed Processing* Referenzmodell der International Organization for Standardization [Int92] definiert 8 Arten von Transparenz. die als Nächstes näher betrachtet werden:

1. **Zugriffstransparenz** verbirgt Unterschiede in der Datendarstellung (z. B. Little/-Big Endian) sowie Unterschiede im Aufruf von Prozeduren.
2. **Ortstransparenz** verbirgt den physikalischen Ort einer Ressource.
3. **Migrationstransparenz** verbirgt die Möglichkeit einer Ressource, zwischen Systemen zu migrieren.
4. **Relokationstransparenz** verbirgt, dass es möglich ist, auf eine Ressource zuzugreifen, während sie migriert.
5. **Fehlertransparenz** verbirgt Fehler, die beim Zugriff auf eine Ressource auftreten können sowie etwaige Gegenmaßnahmen.
6. **Replikationstransparenz** verbirgt die Replikation einzelner (oder aller) Ressourcen.
7. **Persistenztransparenz** maskiert den Aktivierungszustand einer Ressource.
8. **Transaktionstransparenz** verdeckt sämtliche Aktivitäten, die notwendig sind, um die Konsistenz einer Ressource zu gewährleisten (ACID-Eigenschaften).

Festzuhalten bleibt darüber hinaus, dass nicht in jedem verteilten System alle Arten von Transparenz erstrebenswert sind, sondern je nach Kontext nur einige wünschenswert bzw. sinnvoll sind. [TS08] Möchte man bei mobilen Systemen, z. B. in einem Szenario explizit auf den Ort eines Gerätes eingehen, um anhand seiner Position gewisse Empfehlungen (z. B. Auffinden einer Tankstelle) zu treffen, ist Ortstransparenz natürlich keine erstrebenswerte Eigenschaft. Bei der Wahl des Transparenzgrades sollten daher auch Aspekte wie z. B. Kosten, Leistung, Verständlichkeit und Sinnhaftigkeit berücksichtigt werden. [TS08] Auch in Bezug auf die Entwicklung der Managementinfrastruktur und damit auf das Ziel dieser Diplomarbeit ist Transparenz zur Abschirmung von Komplexität ein wichtiger Faktor. So hat in diesem Kontext vor allem die Fehlertransparenz eine sehr hohe Priorität.

Offenheit

Eine weitere Kernanforderung von verteilten Systemen ist die *Offenheit*. Sie beschreibt, wie leicht ein System erweitert und geändert werden kann. [Emm03] Um einen hohen Grad an Offenheit zu erreichen, sollte sich ein verteiltes System an Standards orientieren, d.h. die angebotenen Dienste sollten wohl definierte und wohl dokumentierte Schnittstellen besitzen, um die Syntax und Semantik von Diensten auf einheitliche Weise zu beschreiben. Hierbei ist es sinnvoll, anerkannte und weit verbreitete Standards zu verwenden, um z. B. Abhängigkeiten von Lieferanten zu vermeiden. [Emm03] Ein Beispiel für eine solche erfolgreiche und anerkannte Standardisierung sind die RFC-Dokumente, die die Entwickler der Internetprotokolle Anfang der 80er-Jahre veröffentlicht haben. [CDK05]

Ein weiterer wichtiger Punkt ist, dass Standards bzw. die jeweiligen Schnittstellendefinitionen *richtig* spezifiziert sein sollten, wobei *richtig* bedeutet, dass die Spezifikation vollständig und neutral ist. *Vollständigkeit* verlangt in diesem Kontext, dass sämtliche Informationen vorhanden sind, um eine Implementierung zu ermöglichen, während *neutral* heißt, dass die Spezifikation nicht vorschreibt, wie eine Implementierung auszusehen hat. Beide Eigenschaften verbessern die Interoperabilität und die Portabilität. *Interoperabilität* beschreibt den Grad, nach dem Systeme oder Komponenten, die von unterschiedlichen Herstellern entwickelt wurden, zusammenarbeiten können. *Portabilität* beschreibt, wie gut eine Anwendung, die für ein bestimmtes System entwickelt wurde, ohne Anpassung auf ein anderes System portiert werden kann. [TS08]

Skalierbarkeit

> „*A system is described as scalable if it will remain effective when there is significiant increase in the number of resources and the number of users.*“ [CDK05, S. 19]

Nach *Neumann* [Neu94] werden 3 Dimensionen von Skalierbarkeit unterschieden:

- **Numerisch:** Numerische Skalierbarkeit beschreibt, wie gut es möglich ist, einem System weitere Komponenten oder Benutzer hinzuzufügen.
- **Geografisch:** Diese Dimension verdeutlicht die mögliche geografische Ausdehnung eines Systems.
- **Administrativ:** Ein System gilt als administrativ skalierbar, wenn es trotz steigender Anzahl unabhängiger beteiligter Organisationen effektiv und leicht zu handhaben bleibt.

Der Grad der Skalierung beeinflusst die Zuverlässigkeit, Last, Performanz und Heterogenität eines Systems. Eine starke numerische Skalierung (also eine große Anzahl an

Computern innerhalb des Systems) führt unweigerlich zu einem höheren Verkehrsaufkommen und erhöht damit die Wahrscheinlichkeit, dass einzelne Teile des Systems überlastet werden. Die geografische Ausdehnung eines Systems hat Einfluss auf Latenzzeiten, außerdem sinkt bei hoher geografischer Skalierung die Wahrscheinlichkeit, dass alle Komponenten innerhalb des Systems miteinander kommunizieren können. Des Weiteren beeinflusst die Skalierung die Administration eines Systems. Große Systeme unterliegen zumeist größeren Änderungen, sodass die Administration, vor allem dann, wenn sie sich über mehrere Organisationen mit unterschiedlichen internen Richtlinien erstreckt, erschwert wird. Weiterführend lässt sich festhalten, dass eine hohe Skalierung zu höherer Heterogenität in Bezug auf die Hard- und Software führt, was die Administration ebenfalls erschwert.

Um die genannten Probleme zu vermeiden, gibt es mehrere Skalierungstechniken wie beispielsweise die Replikation oder Verteilung. Bei ersterer werden Ressourcen oder Dienste repliziert, um die Last auf mehrere Computer zu verteilen und die Verfügbarkeit bzw. Zuverlässigkeit zu erhöhen. Letzteres bringt die gleichen positiven Effekte, indem eine Komponente zerlegt und über das gesamte System verteilt wird. [Neu94]

2.2 Mobile Systeme

Nachdem im vorangegangenen Kapitel die allgemeinen Anforderungen und Ziele von verteilten Systemen vorgestellt wurden, behandelt der folgende Teil eine spezielle Klasse von verteilten Systemen, die so genannten mobilen Systeme. Nach einer kurzen Einführung in das Gebiet der mobilen Systeme wird hier zunächst vorgestellt, was Mobilität überhaupt genau bedeutet und welche Aspekte von Mobilität sich unterscheiden lassen. Im folgenden Part werden mobile Systeme kategorisiert und die genauen Unterschiede und Restriktionen im Vergleich zu stationären Systemen verdeutlicht. Des Weiteren werden Arten von mobilen Endgeräten, der drahtlosen Kommunikation sowie Ad-hoc-Netzwerke vorgestellt.

2.2.1 Einführung

Bereits Mitte der 60er-Jahre prognostizierte *Gordan Moore* mit seinem berühmten Gesetz, dass sich die Leistungsfähigkeit von Prozessoren etwa alle 18 Monate verdoppelt. [Moo65] Dieses Gesetz, das als Faustregel und nicht als wissenschaftliches Gesetz zu verstehen ist, hat bis in die heutige Zeit mehr oder weniger präzise Gültigkeit und wird nach Ansicht von Experten auch noch einige Zeit Bestand haben. Neben den Errungenschaften in der Mikroelektronik wurden und werden darüber hinaus aber auch in anderen Bereichen wie Netzwerktechnologie und Materialwissenschaften sowie bei wichtigen technologischen Parametern wie Kommunikationsbandbreite und Speicherkapazität stetig Verbesserungen erzielt. [Mat03] Bereits Anfang der 80er-Jahre konnten erstmals tragbare Computer hergestellt werden, die zudem die Möglichkeit hatten via Modem, mit ande-

ren Rechnern zu kommunizieren. Das Paradigma des *Mobile Computings* war geboren. [CDK05] Fast 30 Jahre später sind tragbare Computer, Mobiltelefone und andere mobile Endgeräte weit verbreitet und kaum noch aus dem Alltag wegzudenken, weshalb geeignete Infrastrukturen und Lösungen für die Kooperation zwischen Teilnehmern und Komponenten in mobilen Systemen von zentraler Bedeutung sind.

2.2.2 Arten der Mobilität

Unter dem Adjektiv „mobil“ versteht man so viel wie „beweglich“ oder auch „nicht an einen festen Standort gebunden“. Das Wort hat seine Herkunft in dem französischen Wort „mobile“ (im Sinne von beweglich, einsatzbereit) und entstammt der Militärsprache des 18. Jahrhunderts. [AW01]

- **Endgerätemobilität (Terminal Mobility):** Unter Endgerätemobilität versteht man die Fähigkeit eines Endgeräts, die Verbindung zu einem Netzwerk aufrechtzuerhalten, während es seinen physikalischen Ort ändert und zwar ohne Unterbrechung der jeweils genutzten Dienste. Als Beispiel für solch eine Art von Mobilität lässt sich die Nutzung eines Mobiltelefons aufführen, das seine Verbindung beibehalten kann, obwohl es über große geografische Distanzen und damit durch unterschiedliche Mobilfunkzellen bewegt wird. [WK01]

- **Benutzermobilität (Personal Mobility):** Die Möglichkeit eines Benutzers, mehrere Endgeräte zu verwenden, wird auch als Benutzermobilität bezeichnet. Um diese Art von Mobilität umzusetzen, sind eindeutige Identifikatoren vonnöten, damit Endgeräte einem Benutzer zugeordnet werden können. [WK01]

- **Dienstmobilität (Service Mobility):** Ist die Nutzung eines Dienstes über unterschiedliche Endgeräte und Netzwerke möglich, so spricht man von Dienstmobilität. [WK01] Ein Dienst mit einer solchen Charakteristik ist beispielsweise das „World Wide Web“, mit dessen Hilfe der Austausch von Multimedia-Dokumenten im Internet ermöglicht wird.

- **Sitzungsmobilität (Session Mobility):** Diese Art von Mobilität erlaubt den Wechsel eines Endgerätes bei gleichzeitiger fortgesetzter Nutzung des aktuellen Datenbestandes. [Str04] Hierbei lässt sich zwischen diskreter und kontinuierlicher Sitzungsmobilität unterscheiden. Die erste Kategorie erlaubt das Transferieren einer Sitzung zwischen zwei Endgeräten nur zu bestimmten Synchronisationszeitpunkten, die Letztgenannte erlaubt zu jedem Zeitpunkt einen Transfer. [Hes05] Ein Beispielszenario wäre hier, dass eine Person ein Gespräch zuhause per Festnetztelefon annimmt und dieses Gespräch, ohne die Sitzung zu beenden (also ohne den Anruf zu unterbrechen), auf das Mobiltelefon überträgt, um außerhalb des Gebäudes weiter zu telefonieren.

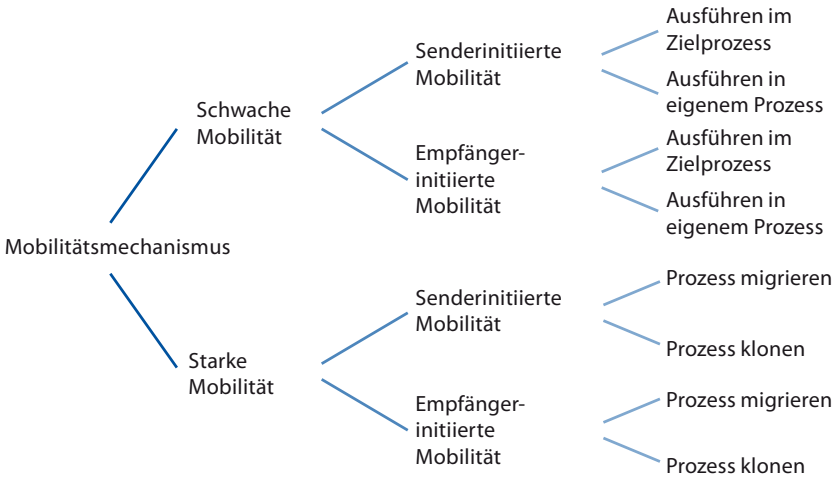

Abbildung 2.1: Möglichkeiten der Codemigration (aus [TS08])

- **Codemobilität (Code Mobility):** Ein Codesegment wird als mobil bezeichnet, sofern es die Möglichkeit hat, innerhalb der Netzwerktopologie zwischen Knoten zu migrieren. Es wird hier zwischen *starker* und *schwacher* Codemobilität unterschieden. Die erste Art beschreibt eine Art von Migration, bei der sowohl der Code als auch der zugehörige Ausführungszustand mit übertragen werden, während bei letzterer Art lediglich der Code und nicht der Ausführungszustand transferiert wird. Bei schwacher Mobilität muss ein Programm nach dem Transfer folglich wieder von seinem Ausgangszustand aus gestartet werden, wie dies zum Beispiel bei Java-Applets der Fall ist. Die kompliziertere, aber auch leistungsfähigere Variante ist die starke Mobilität, bei der der Zustand der Ausführung nicht verloren geht, was z. B. bei mobilen Agenten angestrebt wird. Eine weitere Unterscheidung lässt sich danach treffen, ob die Migration vom Sender (also dem Knoten, auf dem sich der Code aktuell befindet) oder vom Empfänger (also der Zielmaschine aus) initiiert wird. Starke Mobilität kann darüber hinaus auch realisiert werden, indem das jeweilige Codesegment geklont wird, d.h., nach der Übertragung sowohl auf dem Sender- als auch auf dem Empfängerknoten parallel zur Ausführung kommt. [FPV98] [TS08] Abbildung 2.1 verdeutlicht die einzelnen Umsetzungsarten.

2.2.3 Paradigmen mobiler Systeme

Es lassen sich folgende Paradigmen unterscheiden:

- **Mobile Computing:** Unter diesem Oberbegriff wird der Einsatz von tragbaren Computern verstanden, die mit anderen Geräten drahtlos kommunizieren können. [FZ94]
- **Nomadic Computing:** ist ein von *Leonard Kleinrock* geprägter Begriff. Unter „Nomadic“ wird der Wechsel von Individuen zwischen verschiedenen Orten wie beispielsweise dem Zuhause, dem Arbeitsplatz oder Hotels verstanden. „Computing“

bezeichnet in diesem Kontext Dienste, die den jeweiligen nomadischen Individuen über unterschiedliche Endgeräte (PDA, Laptop, Workstation, Handy) zur Verfügung stehen. [CDK05] Im Kern wird unter Nomadic Computing demnach die periodische drahtlose oder drahtgebundene Nutzung von Diensten an verschiedenen Orten mit unterschiedlichen Geräten verstanden („anytime, anywhere“). Der einfache Wechsel zwischen dem drahtgebundenen Rechner am Arbeitsplatz und dem Rechner zuhause ist hierbei ebenfalls als nomadische Bewegung zu verstehen, was verdeutlicht, dass drahtlose Kommunikation keineswegs obligatorisch ist. Des Weiteren ist das permanente Bestehen einer Netzwerkverbindung ebenfalls nicht zwingend erforderlich. [Kle95] Eine Architektur für Nomadic Computing sollte für den Benutzer trotz potenziell häufiger Änderungen in Bezug auf Orte, Umgebungen, Endgeräte, Bandbreite, Latenzzeiten etc. möglichst transparent sein. Ein einfaches Anwendungsbeispiel für Nomadic Computing ist ein Vertreter, der von unterschiedlichen Orten aus auf sein Unternehmensnetzwerk zugreift und damit orts- und geräteunabhängig Zugriff auf die gleichen Dienste hat. [ZN06] [Kle96]

- **Ubiquitous Computing/ Pervasive Computing/ Ambient Intelligence:** Der Begriff **Ubiquitous Computing** wurde 1988 von *Mark Weiser* geprägt. [ML01] In seinem Aufsatz „The Computer for the 21st Century“ [Wei95] sagt *Weiser* voraus, dass Computer in naher Zukunft allgegenwärtig sein werden. In seiner Vision werden Computer nicht nur sehr klein, sondern völlig unsichtbar, z. B. in Gegenständen (auch „smarte“ oder intelligente Gegenstände genannt), sein und uns bei unseren täglichen Aufgaben unterstützen. Nach *Weiser* werden diese Computer in einem allgegenwärtigen Netzwerk miteinander vernetzt sein und darüber hinaus die Fähigkeit haben, ihre Umgebung wahrzunehmen, um eine effiziente Unterstützung zu ermöglichen (siehe auch *Context-Aware Computing*). Ubiquitous Computing beschreibt folglich eine idealistische, humanzentrierte Technikvision, bei der Computer oder allgemein die Technik als reines Mittel zum Zweck in den Hintergrund tritt, damit man sich auf sein eigentliches Ziel konzentrieren kann. [LM03] Der Begriff des **Pervasive Computing** wurde von der Industrie geprägt und bezeichnet im Grunde genommen ebenfalls die allgegenwärtige Informationsverarbeitung, allerdings mit dem Ziel, auf Basis bereits vorhandener Mobile Computing-Technologien diese schon kurzfristig verfügbar zu machen. **Ambient Intelligence** ist der europäische Begriff für diese Art von Informationsverarbeitung, wobei hier zusätzlich Aspekte wie Mensch-Maschine-Interaktion sowie künstlicher Intelligenz berücksichtigt werden. Alle drei Begriffe streben also zumindest im Kern das Ziel an, die Menschen und wirtschaftliche Prozesse mithilfe einer großen Anzahl an Mikroprozessoren und Sensoren zu unterstützen, weshalb ihre Unterscheidung nach *Mattern* auch als „eher akademisch“ klassifiziert wird. [LM03]

Abbildung 2.2 verdeutlicht die Unterschiede der einzelnen Gebiete anhand des Grades der Einbettung und des Grades der Mobilität. Neben den genannten Arten existieren

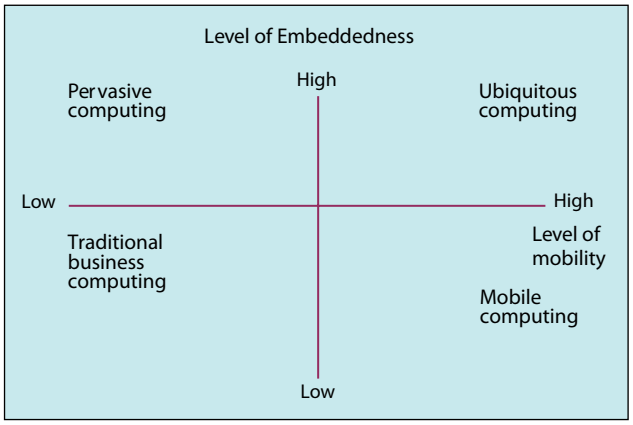

Abbildung 2.2: Klassifizierung der Paradigmen (aus [LY02])

darüber hinaus auch noch weitere Teilfelder des Mobile - und Ubiquitous Computing wie z. B. wearable - , tangible computing und augmented reality [CDK05], auf die hier aber nicht näher eingegangen wird.

2.2.4 Restriktionen von mobilen Systemen im Vergleich zu stationären Systemen

Bei der Entwicklung von Infrastrukturen für mobile Systeme müssen spezielle Randbedingungen, die aus der Mobilität der Endgeräte resultieren, berücksichtigt werden:

- **Endliche Energiequelle:** Aufgrund ihrer Mobilität können mobile Endgeräte nicht permanent an ein festes Stromnetzwerk angeschlossen werden und sind daher auf Batterien und Akkus angewiesen. Damit diese Energiequelle möglichst lange ohne erneute Aufladung hält, muss sowohl bei der Hardware als auch bei der Software darauf geachtet werden, dass möglichst wenig Energie verbraucht wird. Um einen hohen Grad an Mobilität zu erhalten, ist darüber hinaus die ebenfalls recht energieintensive drahtlose Kommunikation vonnöten, weshalb auch hier auf eine ökonomische Nutzung geachtet werden sollte. Die begrenzte Energiequelle erhöht darüber hinaus die Wahrscheinlichkeit eines Fehlers, da die vollständige Entladung des Akkus bzw. der Batterie unweigerlich zum Abbruch aller laufenden Anwendungen führt. [CDK05]

- **Geringere Kapazitäten:** Geschwindigkeit, Speichergröße, Displaygröße und auch Netzwerkbandbreite sind bei mobilen Endgeräten im Vergleich zu stationären Rechnern deutlich geringer. Zwar führt die Forschung zu permanent besseren und leistungsfähigeren mobilen Endgeräten, allerdings werden diese im Vergleich zu den stationären Geräten immer etwas weniger Leistung bieten. Dieser Unterschied wird sowohl durch die beschränkte Größe und das Gewicht, das bei einem mobilen Gerät um die Portabilität zu gewährleisten oder sie in Gegenständen einzubetten klein

gehalten werden muss hervorgerufen, als auch durch den zuvor erwähnten begrenzten Energiehaushalt. Eine Erhöhung z. B. der Prozessorgeschwindigkeit führt dementsprechend zu einer Erhöhung des Energieverbrauchs, während die Größe des Gerätes z. B. die Anzahl der Transistoren begrenzt. [CDK05]

- **Allgemeine Risiken:** Bei mobilen Endgeräten besteht im Gegensatz zu stationären Geräten eine höhere Gefahr des Diebstahls, des Verlusts oder der Beschädigung. Diese höhere Wahrscheinlichkeit resultiert direkt aus der Portabilität, da auf diese Weise das jeweilige Gerät leichter herunterfallen oder verloren gehen kann. Des Weiteren ist es wahrscheinlicher, dass einer Person beispielsweise auf der Straße ein Mobiltelefon gestohlen wird als eine komplette Workstation, die sich hinter verschlossenen Türen in einem Büro befindet. [Sat96]

- **Volatile Konnektivität:** Die Fehleranfälligkeit ist bei drahtloser Kommunikation deutlich höher als bei drahtgebundener. Durch die Beweglichkeit der einzelnen Knoten besteht daher zum einen die Gefahr, dass eine Verbindung aufgrund zu hoher Entfernung zwischen zwei Kommunikationspartnern unterbrochen wird oder z. B. Gebäude die Verbindung verschlechtern oder ganz abbrechen lassen. Selbst bei drahtloser Kommunikation zwischen zwei stationären Kommunikationspartnern können immer noch Autos oder Personen, die sich zwischen ihnen bewegen, zu Verbindungsabbrüchen oder stark schwankender Bandbreite und - Latenzzeiten führen. [CDK05] Des Weiteren wird die Bandbreite bei drahtloser Kommunikation zwischen den einzelnen Benutzern innerhalb einer Funkzelle aufgeteilt, weshalb vor allem bei stark volatilem Nutzeraufkommen innerhalb einer Zelle mit großen Schwankungen in Bezug auf diesen Parameter zu rechnen ist. [FZ94]

- **Sicherheitsaspekte:** Die Verwendung von drahtloser Kommunikation birgt darüber hinaus einige Sicherheitsrisiken. So kann die drahtlose Interaktion zwischen zwei Kommunikationspartnern z. B. deutlich leichter abgehört werden. Um derartigen Lauschangriffen entgegenzuwirken, ist folglich Verschlüsselung vonnöten. Als problematisch erweist sich in diesem Kontext allerdings, dass eine Verschlüsselung ebenfalls sehr energie- und rechenaufwändig ist. Erschwerend kommt noch hinzu, dass konventionelle Sicherheitsprotokolle häufig Annahmen im Hinblick auf Endgeräte und Konnektivität machen, die in volatilen Systemen nicht erfüllt werden. So sollte sich ein Sicherheitssystem für mobile Systeme beispielsweise nicht auf die Integrität einer Untermenge von Endgeräten verlassen, wenn diese z. B. leicht durch Diebstahl kompromittiert werden können. [CDK05]

2.2.5 Kategorien mobiler Endgeräte

Um einen Überblick über die Vielfalt an mobilen Endgeräten und ihrer Funktionalität zu bekommen, werden im Folgenden einige Arten betrachtet:

- **Handheld Computer** sind internetfähige mobile Geräte, die hauptsächlich für Bürofunktionalität genutzt werden. Wie der Name bereits verdeutlicht, sind diese Geräte sehr klein, leicht und können daher problemlos zum Beispiel in der Hosentasche transportiert werden. Je nach Ausführung beinhaltet ein solches Gerät eine Tastatur und/oder einen Touchscreen, der mithilfe eines speziellen Stiftes bedient wird. Geräte ohne Tastatur werden auch *Personal Digital Asisstent* (PDA) genannt. [Han03] Neben Consumer-Modellen existieren auch so genannte industrietaugliche Modelle, bei denen gewisse Charakteristika für den Einsatz im Industriebereich modifiziert wurden (z. B. Schutzmechanismen, um Beeinträchtigung durch Staub oder Feuchtigkeit zu verhindern).

- **Mobiltelefon:** Unter einem Mobiltelefon versteht man ein mobiles Gerät, mit dessen Hilfe ortsungebunden telefoniert werden kann. Neben der eigentlichen Telefonfunktionalität besitzen aktuelle Mobiltelefone eine Vielzahl an weiteren Funktionen wie z. B. die Möglichkeit Spiele zu spielen, Radio zu hören, Kurznachrichten austauschen und auch im Internet zu surfen. In Bezug auf die Funktionalität herrscht folglich eine sehr hohe Vielfalt, dies lässt sich auch in Bezug auf die dazu gehörenden Betriebssysteme sagen. [Han03]

- **Smart Phones** sind eine Kombination aus Mobiltelefon und Handheld Computer. Generell unterstützen Smartphones folglich Telefonie, Datendienste und Computeranwendungen.

- **Notebooks** sind im Grunde genommen tragbare Personalcomputer. Sie besitzen eine Tastatur, einen Bildschirm und ähnliche Hardware, allerdings muss, um Portabilität zu gewährleisten, alles etwas kleiner ausfallen. Die Mobilität wird demnach zwar unter anderem durch Leistungsverlust erkauft, allerdings sind heutige Notebooks bereits so leistungsfähig, dass selbst grafisch aufwändige Spiele und andere rechenaufwändige Anwendungen mit ihnen durchgeführt werden können. Als Betriebssysteme lassen sich bei diesen Geräten die üblichen Desktop-Betriebssysteme verwenden und damit auch die zugehörigen Anwendungen ausführen.

- **Subnotebooks** lassen sich in Bezug auf Größe, Rechenpower und Funktionalität in etwa zwischen Handhelds und normalen Notebooks ansiedeln. Vorteile gegenüber einem normalen Notebook sind weniger Gewicht und Größe und damit bessere Portabilität sowie eine längere Akkulaufzeit, die unter anderem durch die geringere Leistung erkauft wird. Häufig wird aus Platzgründen bei diesen Geräten auf CD-ROM - und Floppy Laufwerke verzichtet. Da Subnotebooks aber immer noch zu groß sind, um sie in der Hosentasche zu verwahren und weniger Funktionalität als ihre größeren Pendants aufweisen, sind sie in der Popularität deutlich unter Notebooks anzusiedeln. [Han03]

- Ein **Tablet PC** lässt sich als Mischform eines Handheld Computers und eines Note-

books verstehen. Es besteht aus einem Bildschirm, der etwa die Größe eines DIN-A4-Blattes umfasst, wobei völlig auf eine Tastatur verzichtet wird. Die Bedienung dieses Geräts läuft ausschließlich über den bedienungssensitiven Bildschirm mithilfe eines speziellen Stiftes. Bezüglich der Leistung kommt es nicht ganz an ein Notebook heran, ist einem Handheld Computer allerdings überlegen. Der Nachteil, dass keine Tastatur vorhanden ist, wird zumindest zum Teil durch eine gute Handschriftenerkennung, die angepasste Betriebssysteme wie z. B. Windows XP Tablet PC beinhalten, abgeschwächt.

2.2.6 Drahtlose Kommunikation

Wie es der Name bereits verdeutlicht, versteht man unter „drahtloser Kommunikation" eine Art von Signalübertragung, bei der man nicht auf einen Leiter zurückgreift, also z. B. einen Draht. Um Signale ohne jegliche Führung durch einen „Raum" zu übermitteln, werden elektromagnetische Wellen (u.a. Radiowellen, Mikrowellen, Licht) verwendet, wobei sich die einzelnen Arten lediglich in der Frequenz unterscheiden. [Sch00] Der folgende Abschnitt behandelt einige der gängigen drahtlosen Technologien, wobei das Klassifikationskriterium die räumliche Ausdehnung ist.

- **Wireless Wide Area Network:** Diese Netzwerke sind das drahtlose Pendant zu Wide Area Networks. Die Ausbreitung dieser Netze umspannt sehr große geografische Entfernungen wie zum Beispiel Länder oder sogar Kontinente [Tan03]. Die wohl prominentesten Vertreter dieser Kategorie sind die Mobilfunknetze bzw. die Mobilkommunikation. Unter *Mobilfunk* wird heute eine Form der Telekommunikation verstanden, bei der ein Provider (Dienstanbieter) es ermöglicht, Sprache und Daten zwischen mobilen Endgeräten (den Teilnehmern) auszutauschen. Im Hinblick auf die Technologie lassen sich drei Entwicklungsstufen (auch Generationen genannt) unterscheiden (vgl. Abbildung 2.3): Mit der ersten Generation sind analoge Systeme wie z. B. das *C-Netz* in Deutschland oder der *Advanced Mobile Phone Service* (AMPS) gemeint. Die zweite Generation bezeichnet digitale Systeme, zu denen der Standard *Global System for Mobile Communications* (GSM) gehört, der weltweit am weitesten verbreitet ist und eine Datenübertragungsrate von bis zu 9,6 kbit/s ermöglicht. Die dritte Generation soll digitale Kommunikation über den Standard *Universal Mobile Telecommunications System* (UMTS) verwirklichen, der hohe Datenübertragungsraten (bis zu 2Mbit/s) sowie schnelle Internetverbindungen ermöglicht und damit auch den komfortablen Transport von multimedialen Daten erlaubt. [Sch00]

- **Wireless Local Area Network:** Die Reichweite dieser lokalen Netze beträgt je nach räumlichen Verhältnissen 30-300 Meter. *Wireless Local Area Network* (WLAN) wird zum Teil als Oberbegriff für jede drahtlose lokale Vernetzung verwendet, WLAN bezeichnet aber eigentlich die *IEEE 802.11* Spezifikation, die in der Fassung *802.11g*

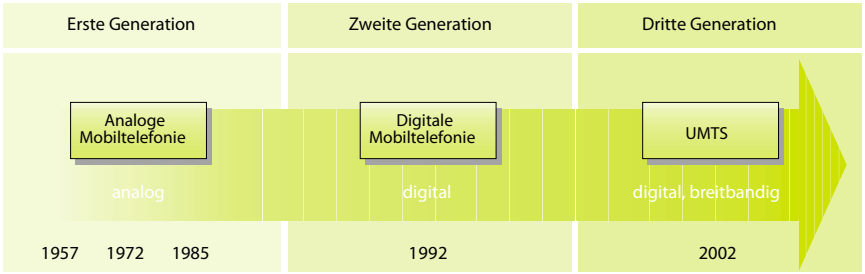

Abbildung 2.3: Die drei Generationen des Mobilfunks (aus [Eri01])

Datenraten von bis zu 54MBit/s ermöglicht. Der WLAN-Standard beinhaltet zwei Vernetzungsmodi, einen Infrastrukturmodus, bei dem die Kommunikation über einen Access Point geregelt wird und einen Ad-hoc-Modus, der eine Peer-to-Peer-Vernetzung ermöglicht. Typische Anwendungsbereiche sind Heim- oder Firmennetzwerke. [TP04]

- **Wireless Personal Area Network:** Die Ausbreitung dieser Netze umfasst lediglich wenige Zentimeter bis zu ein paar Metern. *Wireless Personal Area Networks* (WPAN) werden hauptsächlich genutzt, um komfortabel Endgeräte miteinander oder mit Peripheriegeräten in der Umgebung wie z. B. Druckern zu vernetzen. Die bekanntesten Standards auf diesem Gebiet sind *Bluetooth* und *Infrared Data Association* (IrDa). Der weit verbreitete Standard Bluetooth, der heute in einer Vielzahl an Endgeräten (Handy, PDA, Headset) verwendet wird, arbeitet wie auch WLAN im lizenzfreien *ISM-Band* (2400-2483,5 MHZ) und kann daher zulassungsfrei betrieben werden. Die Reichweite beträgt standardmäßig 10 Meter, wobei eine Datenübertragungsrate von bis zu 1 Mbit/s erreicht werden kann. Zu bemerken bleibt allerdings, dass bereits Techniken existieren, um noch größere Datenübertragungsraten und Reichweiten zu erreichen. IrDa ist ein Standard zur Datenübertragung mittels Infrarot, was den Vorteil hat, dass diese Technologie im Gegensatz zur Funktechnik gegen elektromagnetische Einflüsse unempfindlich ist, dafür allerdings durch andere Lichtquellen oder Reflexion gestört werden kann. Die Datenübertragungsrate beträgt bei einer Reichweite von 1-2 Metern beim Standard *IrDa 1.2* bis zu 16 Mbit/s (*Very Fast Infrared*, VFIR). [TP04]

2.2.7 Ad-hoc-Netze

Unter einem Ad-hoc-Netz versteht man eine Netzvariante, bei der keine Infrastruktur, also insbesondere kein Access Point oder keine Basisstation notwendig ist. Diese Netzart ermöglicht die spontane Interaktion zwischen (mobilen) Endgeräten und ist damit gerade in Gebieten oder Regionen geeignet, wo eine adäquate Infrastruktur zu teuer, komplex, zeitaufwändig oder gar nicht möglich ist. [Sch00] Sofern die einzelnen Knoten die Möglichkeit haben, sich selbst zu bewegen oder von den jeweiligen Benutzern

„bewegt“ werden, spricht man auch von *mobilen Ad-hoc-Netzen* (auch *MANETs* genannt). Befindet sich jeder Knoten im Netz mit jedem anderen Knoten in direkter Kommunikationsreichweite, so handelt es sich um ein *single-hop Ad-hoc-Netz*. Da dies in der Regel nicht der Fall ist, sind gerade mobile *multi-hop Ad-hoc-Netze*, bei denen zwischen Quelle und Ziel potenziell mehrere Zwischenknoten, die als Router fungieren müssen, Gegenstand intensiver Forschung und werden im Folgenden näher betrachtet.

Das Charakteristikum der Knotenmobilität führt zu diversen Problemen und Herausforderungen, die bei der Entwicklung von Middleware für MANETs berücksichtigt werden müssen. So unterliegt die Topologie in diesen Netzen potenziell ständigen und unvorhersehbaren Änderungen, weil neue Geräte im Netz auftauchen, alte das Netz verlassen oder einfach in Bewegung sind. Weitere Probleme ergeben sich aus der Verwendung drahtloser Kommunikation und aus den mannigfaltigen Beschränkungen, denen mobile Endgeräte allgemein unterliegen (vgl. 2.2.4).

Aufgrund der genannten Unterschiede und Restriktionen sind die Algorithmen und Ansätze von Netzen mit fester Infrastruktur und/oder stationären Knoten nicht anwendbar, weshalb Anpassungen notwendig sind, die diese spezifischen Unterschiede berücksichtigen. Da keine Infrastruktur vorhanden ist, müssen das Netz bzw. die einzelnen Teilnehmer selbst die Fähigkeit besitzen, sich automatisch zu konfigurieren und zu verwalten. Um die zuletzt genannte Anforderung zu ermöglichen, sind sowohl spezielle Mechanismen zur Adressierung der einzelnen Knoten als auch adäquate Routing-Algorithmen notwendig, damit jeder Knoten die Möglichkeit hat, mit jedem anderen Knoten zu kommunizieren. [Gün04]

Adressierung

Um Knoten voneinander zu unterscheiden und zu identifizieren, ist eine adäquate Adressierung notwendig. Die Adressierung erfordert einheitliche Adressen sowie ein Verfahren, das den Knoten eindeutige Adressen zuweist. Für die erste Anforderung ist prinzipiell jedes Schema sinnvoll, das für das jeweilige Anwendungsgebiet einen ausreichend großen Adressraum bietet. Geeignet ist zum Beispiel das Adressierungsschema des Internets, also die Verwendung von IP-Adressen zur Identifizierung von Knoten. Im Folgenden wird exemplarisch eines der bekannten Verfahren zur Verteilung von eindeutigen IP-Adressen beschrieben. [PRD00]

Jeder neue Knoten in einem Ad-hoc-Netz wählt zunächst zufällig eine IP-Adresse aus dem jeweiligen Adressraum. Um nun herauszufinden, ob die Adresse bereits belegt ist, wählt er eine weitere temporäre IP-Adresse aus einem speziellen Adressraum, unter der er während der Validierung der eigentlichen Adresse erreichbar ist. Im Anschluss sendet der Knoten eine Nachricht mit beiden Adressen an alle erreichbaren Knoten und wartet ein bestimmtes Zeitintervall. Jeder Knoten, der diese Nachricht erhält, vergleicht seine eigene IP mit der zu prüfenden und leitet, sofern diese nicht übereinstimmen, die Nachricht an alle erreichbaren benachbarten Knoten weiter. Sofern ein Knoten die zu prüfende

Adresse bereits selbst verwendet, schickt er eine Nachricht an den ursprünglichen Absender zurück, woraufhin der Initiator eine neue IP-Adresse wählt und das Verfahren von vorne beginnt. Falls das Zeitintervall des Wartens ohne „Einwand“ verstreicht, geht der Knoten davon aus, dass die Adresse nicht vergeben ist, behält diese und gibt die temporäre Adresse wieder frei. Ein Nachteil an diesem Verfahren ist beispielsweise, dass ein bösartiger Knoten jedes Mal behaupten könnte, er habe die jeweilige Adresse und damit leicht den Aufbau eines Ad-hoc-Netzes verhindern kann. Darüber hinaus besteht ein Problem, wenn ein Knoten, der die zu prüfende Adresse bereits gewählt hat, keinen Einwand erhebt, weil er z. B. temporär nicht erreichbar ist und es dadurch zu einer doppelten Vergabe kommt.

Routing

Mit Routing wird in diesem Kontext die Wegeermittlung und Wegewahl im Ad-hoc-Netz bezeichnet. Damit Kooperation innerhalb des MANET zwischen den einzelnen Teilnehmern überhaupt möglich ist, sind effiziente Routing-Algorithmen vonnöten, um gezielt Kommunikationsbeziehungen aufbauen zu können. Hierbei gilt: Je besser und schneller der Routing-Algorithmus arbeitet, desto höher ist auch die Leistungsfähigkeit des Netzes. Dementsprechend ist Routing in Ad-hoc-Netzen seit Jahren Bestand intensiver Forschung, weshalb eine Vielzahl an Routing-Algorithmen existiert, wobei es allerdings keinen Algorithmus gibt, der in jedem Szenario geeignet ist, sondern vielmehr das Anwendungsgebiet und die Rand- und Rahmenbedingungen darüber entscheiden, ob ein Algorithmus eingesetzt werden kann oder sollte. Routing-Algorithmen können entweder *statisch* oder *adaptiv* sein. Da MANETs per Definition Knotenmobilität und sich verändernde Topologien beinhalten, sind statische, sich nicht anpassende Ansätze in dieser Umgebung meistens unbrauchbar und werden daher nicht näher betrachtet.

Eine mögliche Klassifikation von adaptiven Algorithmen lässt sich danach aufstellen, ob sie *topologiebasiert* oder *positionsbasiert* sind. Bei der ersten Variante werden Routingentscheidungen anhand von Informationen über die logische Anordnung der Knoten innerhalb des Netzwerkes getroffen. Topologiebasierte Verfahren lassen sich ferner danach klassifizieren, ob sie *reaktiv*, *proaktiv* oder *hybrid* sind. Bei reaktiven Verfahren werden die Routen lediglich „on-demand“, also erst dann, wenn eine Kommunikation zwischen zwei Knoten erforderlich ist, ermittelt. Proaktive Verfahren arbeiten mit Tabellen, in denen die Routing-Informationen gespeichert werden, d.h., jeder Knoten führt eine solche Tabelle und aktualisiert diese periodisch, um den Änderungen bzw. allgemein der Mobilität innerhalb des Netzes gerecht zu werden.

Bei proaktiven Verfahren kann ein Weg mithilfe der Tabelle folglich deutlich schneller ermittelt werden, allerdings wird auch ein großer Anteil der Netzwerkkapazität dafür benötigt, die Tabellen aktuell zu halten. Reaktive Verfahren sind zwar im Prinzip langsamer, aber konsumieren deutlich weniger Bandbreite. Untersuchungen haben gezeigt, dass proaktive Verfahren aufgrund des inhärenten Overheads und der Tatsache, dass ein-

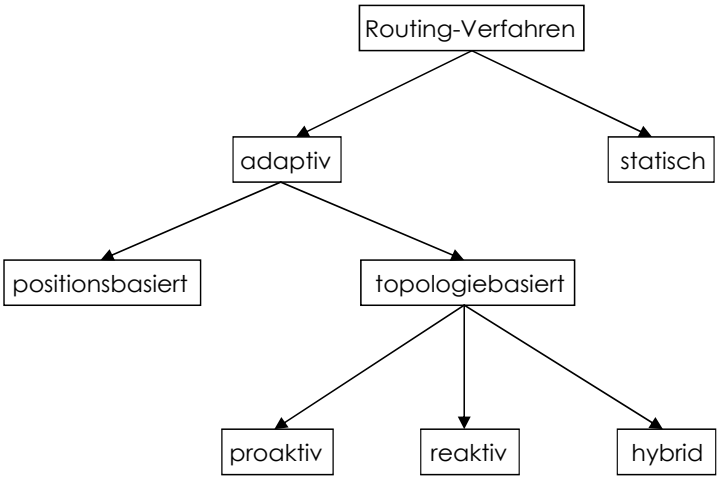

Abbildung 2.4: Klassifikation von Routingverfahren (nach [MCA06])

zelne Einträge in den Tabellen häufig veraltet sind, ein schlechteres Leistungsverhalten haben als reaktive Ansätze. Hybride Algorithmen sind eine Kombination aus den beiden anderen Kategorien mit dem Ziel, einen höheren Grad an Effizienz und Skalierbarkeit zu erreichen, indem beispielsweise für nahe Knoten das proaktive Verfahren und für entfernte ein reaktives Verfahren verwendet wird.

Positionsbasierte Algorithmen versuchen, die Nachteile einiger topologiebasierter Ansätze zu bewältigen, indem zusätzliches Wissen über den physikalischen Ort eines Knotens verwendet wird (z. B. anhand von GPS-Daten), um Datenpakete gezielt an das jeweilige Ziel zu leiten. Hierbei wird die Position des Zielknotens von jedem Zwischenknoten verwendet, um den jeweils nächsten (Zwischen-)Knoten zu bestimmen, wobei der Quellknoten die Zielposition zuvor anhand eines so genannten *Location Service* ermittelt und in das Paket einbettet. Im Vergleich zu topologiebasierten Verfahren hat der positionsbezogene Ansatz den Vorteil, dass Routen zwischen Knoten nicht explizit aufgebaut oder gepflegt werden müssen. Es wird allerdings vorausgesetzt, dass jeder Knoten mit zusätzlicher Hardware ausgestattet ist, um seine eigene Position bestimmen zu können. [MCA06] [Gün04]

Sensornetze

Unter einem Sensornetz versteht man ein spezielles Rechnernetz, dessen Knoten Sensoren sind. Ein Sensorknoten ist hierbei ein physikalisch sehr kleines, batteriebetriebenes Gerät, das geringe Berechnungsfähigkeiten besitzt, drahtlos kommunizieren kann und mit einem Messgerät ausgestattet ist, um physikalische Gegebenheiten zu ermitteln oder zu überwachen. Aufgrund der stetigen technologischen Verbesserungen wird davon ausgegangen, dass es in naher Zukunft Sensoren geben wird, die wenige Kubikmillimeter groß sind und weniger als ein US-Dollar kosten werden. Sensornetze sind folglich eine spezielle Form von Ad-hoc-Netzen, wobei die einzelnen Knoten keine oder lediglich re-

duzierte Mobilität aufweisen. [MCA06] Mit diesen Netzen könnten in naher Zukunft beispielsweise Waldbrände früher entdeckt werden, indem eine Vielzahl an Sensorknoten im jeweiligen Zielgebiet untergebracht wird, die die Temperatur überwachen und ggf. die Feuerwehr benachrichtigen. Weitere Einsatzszenarien sind Verkehrsanalysen, um z. B. Staus früh zu erkennen oder die Analyse von Feindbewegungen im Krieg.

2.3 Kooperation durch mobile Prozesse

Den Schwerpunkt des folgenden Kapitels bilden die *mobilen Prozesse*, die ein wichtiges Kooperationskonzept im Bereich der mobilen Systeme sind. Um die Ziele und Vorteile dieses Konzeptes herauszustellen, werden zunächst das Paradigma und die Charakteristika der serviceorientierten Architektur (*SOA*) vorgestellt. Des Weiteren wird der Begriff der Context-Awareness erläutert und es wird verdeutlicht, warum die Wahrnehmung und Verarbeitung von Kontexten für eine Anwendung im Bereich der mobilen Systeme von Vorteil sind.

2.3.1 Serviceorientierte Architektur

Der folgende Abschnitt verdeutlicht das Konzept und die zugehörigen Charakteristika der serviceorientierten Architektur, deren Umsetzung gerade in mobilen Systemen die Kooperation zwischen den beteiligten Entitäten erleichtert, indem Dienste auf technologieunabhängige und standardisierte Art angeboten, gesucht und genutzt werden können.

Definition

Bevor der eigentliche Begriff definiert wird, werden zunächst die Teilkomponenten *Service* (Dienst) und *Architecture* (Architektur) beschrieben. Unter Services versteht man sich selbst beschreibende, plattformunabhängige Elemente, die eine kostenarme und schnelle Komposition von verteilten Anwendungen erlauben. Ein Dienst kann in diesem Kontext einfache Funktionen bis hin zu komplexen Geschäftsprozessen ausführen und erlaubt einem *Service Provider* (z. B. einer Organisation) die Veröffentlichung und Bereitstellung der eigenen Kernkompetenzen mittels standardisierter Sprachen und Protokolle. [Pap03] Als eine Softwarearchitektur wird eine strukturierte bzw. hierarchische Anordnung von Systemkomponenten inklusive Beschreibung der zugehörigen Beziehungen bezeichnet. [Bal00] Für den Begriff der *serviceorientierten Architektur* (kurz *SOA*) existiert keine einheitliche Definition. Eine für diese Diplomarbeit angemessene Definition lautet wie folgt:

> *„Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“* [Mel07, S:11]

Nach dieser Definition stehen also Aspekte wie Offenheit, Wiederverwendbarkeit, Interoperabilität, Sprachen- und Plattformunabhängigkeit und damit Überwindung von Heterogenität im Fokus. Gleichzeitig wird deutlich, dass es sich bei einer serviceorientierten Architektur nicht um eine spezifische Technik, sondern um eine abstrakte Softwarearchitektur handelt, bei der gewisse wichtige Aspekte betont und von anderen weniger wichtigen abstrahiert wird.

Charakteristika

Ein wichtiger Aspekt der SOA ist die Durchsetzung einer losen Kopplung der Services. Um das zu erreichen, ist eine Bindung zur Laufzeit an die jeweiligen Dienste notwendig, d.h., die Dienste werden nicht statisch zur Programmierzeit festgelegt, sondern bei Bedarf dynamisch ausgewählt. Dies ist gerade in hochdynamischen Umgebungen wie z. B. mobilen Ad-hoc-Netzen wichtig (vgl. 2.2.7), da zum Zeitpunkt der Programmierung eines Prozesses noch gar nicht bekannt ist, welche Services und Serviceanbieter zur Verfügung stehen. Damit diese Art der Bindung überhaupt möglich ist, muss ein Verzeichnis bzw. standardisierter Verzeichnisdienst existieren, in den jeder angebotene Dienst eingetragen wird und in dem eine Anwendung zum gegebenen Zeitpunkt suchen kann, um einen gewünschten Dienst lokalisieren zu können.

Neben der eigentlichen Lokalisierung muss der Dienstnutzer für eine erfolgreiche Interaktion weitere maschinenlesbare Informationen (z. B. verwendete Protokolle, Aufbau der Datenpakete) abrufen können, um den jeweiligen bis dato unbekannten Dienst in Anspruch zu nehmen. Damit dies mit jedem beliebigen Dienst möglich ist, sind offene Standards notwendig. Der Dienstanbieter benötigt also seinerseits eine standardisierte Sprache (*Service Description Language*), um seinen Dienst auf einheitliche implementationsunabhängige Art und Weise zu beschreiben. Die Umsetzung dieser Prinzipien ermöglicht es dem Dienstanbieter, einen Service anzubieten, bei dem die Schnittstelle von der Implementierung getrennt ist und einem Dienstnutzer die automatische und einfache Verwendung und Wiederverwendung von beliebigen Diensten. Abbildung 2.5 verdeutlicht die Schritte der Nutzung eines a priori unbekannten Dienstes.

Neben den genannten Aspekten sollte eine SOA einen hohen Grad an Einfachheit gewährleisten, um eine schnelle Umsetzung zu ermöglichen sowie ein hohes Maß an Sicherheit (u.a. Authentifizierung, Authentisierung) zu gewährleisten. Beide Aspekte sind darüber hinaus Prämissen für eine hohe Akzeptanz einer serviceorientierten Architektur. [Mel07]

WebServices

Eine mögliche Umsetzungsform einer SOA sind *Web Services*. Diese erlauben im Prinzip die automatisierte Maschine-Maschine-Interaktion, indem Dienste auf standardisierte Weise beschrieben, veröffentlicht, gefunden und abgerufen werden können, wobei die bereits genannten Prinzipien und Charakteristika der SOA (also u.a. lose Kopplung)

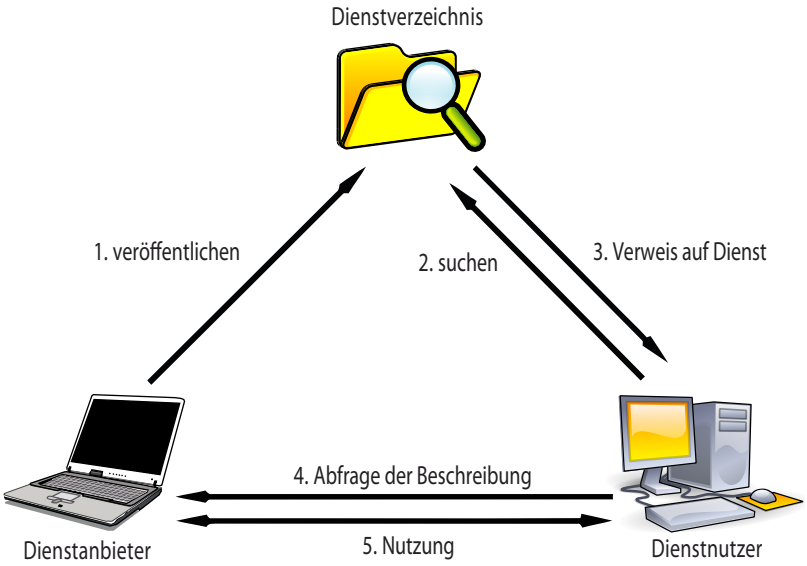

Abbildung 2.5: Rollen und Aktionen einer SOA (nach [Mel07])

angestrebt werden. Auf diese Weise lässt sich also beispielsweise Business-to-Business-Integration bzw. Kooperation über das Internet erreichen.

2.3.2 Context-Awareness

Bei der Kommunikation zwischen Menschen spielt der Kontext, also unter anderem die jeweilige Situation, in der sich zwei Kommunikationspartner befinden, eine wichtige Rolle. So empfinden es andere Leute z. B. als störend, wenn sich Personen während einer Vorführung im Theater oder Kino unterhalten oder ein Telefon klingelt, während das in einer anderen Umgebung (z. B. im Stadion) kein Problem darstellt. Der Kontext Kino/-Theater bzw. Ort macht in diesem Zusammenhang also einen Unterschied aus. Mithilfe des wahrgenommenen Kontextes können sich Menschen also auf die jeweilige Situation einstellen und sich anpassen. Diese Eigenschaften sind auch in der Mensch-Maschine- bzw. Maschine-Maschine-Interaktion erstrebenswert, da eine Anwendung bzw. ein System, das Kontextinformationen sammelt und adäquat verarbeitet, automatisch auf sich ändernde Situationen und Gegebenheiten in der Umgebung reagieren kann, ohne dass ein Benutzer manuell eingreifen muss. Diese Anpassungsfähigkeit ist gerade dann besonders hilfreich, wenn die jeweilige Umgebung, wie zum Beispiel bei mobilen Systemen, sehr dynamisch ist.

Um Anwendungen *context-aware* zu gestalten und von den Vorteilen zu profitieren, ist daher ein tieferes Verständnis dazu notwendig, was Kontext überhaupt ist und was eine solche anpassungsfähige Anwendung ausmacht.

Kontextinformationen

Unter dem Begriff Kontext wird jede Art von Information bezeichnet, die verwendet werden kann, um die Situation einer Entität zu charakterisieren. Entitäten sind in diesem Zusammenhang Personen, Orte oder Objekte, die für die Interaktion zwischen Benutzer und Anwendung als relevant angesehen werden, wobei der Benutzer und die Anwendung ebenfalls berücksichtigt werden. Einige Kontextinformationen sind in der Praxis relevanter als andere, hierzu zählen der Ort, die Identität, die Aktivität und die Zeit. Während der Ort den Raum oder das Gebiet bezeichnet, in dem sich eine Entität aufhält, beinhaltet die Identität Informationen, die es erlauben, Entitäten eindeutig zu unterscheiden. Die Aktivität gibt darüber Auskunft, was eine Entität gerade macht und die Zeit bezieht sich auf Aspekte wie die derzeitige Uhrzeit, Tages- oder Jahreszeit. Mithilfe dieser primären Kontextinformationen für eine Entität lassen sich nun sekundäre Kontextinformationen wie beispielsweise die Anschrift einer Person anhand der Identität ermitteln. [ADB+99]

Kontextsensitive Anwendungen

Für den Begriff *context-awareness* oder seine Synonyme wie *environment-directed*, *context-sensitive* oder *context-adaptive* existieren mannigfaltige Definitionen. Zwei der Definitionen lauten wie folgt:

> *„Eine Anwendung ist kontextbezogen wenn ihr Verhalten durch Kontextinformation beeinflusst wird."* [RBB03, S.3]

> *„A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task."* [ADB+99, S.6]

Während die erste Definition, die von vielen Forschern verwendet wird, eine Anwendung nur dann als kontextsensitiv ansieht, wenn ihr Verhalten anhand der Informationen beeinflusst wird, sind in der zweiten Definition auch Anwendungen kontextbezogen, die diese Informationen z. B. nur darstellen und nicht die Anwendung selbst beeinflussen. Der zuletzt genannte Beschreibungsansatz ist also allgemeiner und umfasst damit ein größeres Anwendungsspektrum. Bei jeder weiteren Erwähnung dieses Begriffs oder seiner Synonyme bezieht sich diese Diplomarbeit daher auf die zweite Definition.

Charakteristika

Nach *Rothermel et al.* lassen sich kontextbezogene Anwendungen wie folgt unterscheiden: [ADB+99]

- **Kontextbezogene Selektion:** Hierunter versteht man die Einbeziehung von Kontextinformationen bei der Auswahl von Diensten oder Informationen. Ein hierzu passendes Szenario ist zum Beispiel das automatische Anzeigen des korrekten

Stadtplanes oder eine Empfehlung im Hinblick auf ein nahe liegendes Restaurant anhand der primären Kontextinformation „Ort“.

- **Kontextbezogene Präsentation** bezeichnet die Fähigkeit, anhand des Kontextes die Darstellung einer Anwendung zu beeinflussen. Die Präsentation einer Karte könnte zum Beispiel anhand der Geschwindigkeit eines Benutzers verändert werden, da eine laufende Person mit einer hoch detaillierten Karte weniger als mit einfachen Richtungsangaben anfangen kann.
- **Kontextbezogene Aktionen** sind Aktionen, die ausgeführt werden, sofern eine bestimmte Kombination an Kontextinformationen auftritt. Denkbar wäre beispielsweise, dass sich ein Handy automatisch abschaltet oder auf Vibrationsalarm wechselt, sofern sich sein Besitzer im Kino befindet und der Film beginnt.

2.3.3 Mobile Prozesse

Das Konzept der mobilen Prozesse wurde im Fachbereich Verteilte Softwaresysteme an der Universität Hamburg entwickelt. Die Idee hinter diesem Konzept besteht darin, potenziell langlebige und vielschichtige (Geschäfts-) Prozesse effizienter abzuwickeln, indem die Migration von Prozessen zwischen Endgeräten unter Beibehaltung des aktuellen Ausführungszustandes ermöglicht wird.

> *„Ein Mobiler Prozess ist eine Abfolge zusammengehöriger (z. T. entfernter) Dienste, deren vollständige Bearbeitung i. Allg. eine längere Zeitspanne andauert und sich dabei durch Migration auch über mehrere Geräte erstrecken kann. Die Ergebnisse eines derartigen Prozesses bestehen aus der Summe aller Effekte, die der Initiator im Zuge der Bearbeitung des Gesamtprozesses erwartet.“* [Kun08, S.141]

Im Gegensatz zu konventionellen Geschäftsprozessen sind mobile Prozesse benutzerzentriert, d.h., sie werden von Benutzern initiiert und in Kooperation mit anderen Endgeräten bzw. Benutzern bearbeitet. Als Anwendungsgebiet sind hierbei nicht nur stationäre Systeme von Relevanz, sondern es stehen gerade hochdynamische, infrastrukturlose Systeme wie z. B. mobile Ad-hoc-Netze (vgl. 2.2.7) im Fokus, um auch in diesen volatilen Systemen effiziente Kooperationen zu ermöglichen und damit einen Schritt in Richtung Ubiquitous Computing (vgl. 2.2.3) zu gehen. Im Hinblick auf die beschriebenen Arten der Codemobilität (vgl. 2.2.2) lässt sich das Konzept der mobilen Prozesse unter dem Begriff der starken Codemobilität kategorisieren. Die Codemigration hat den Vorteil, dass gerade in mobilen Systemen, in denen die einzelnen Teilnehmer bzw. teilnehmenden Geräte leistungstechnisch mannigfaltigen Restriktionen unterliegen (vgl. 2.2.4), das Potenzial benachbarter Endgeräte genutzt werden kann, um die Güte der Kooperation und damit auch die Prozessbearbeitung zu verbessern. [KZL06]

Für den konkreten Einsatz dieser Prozesse sind eine Spezifikationssprache zur Beschreibung bzw. formalen Definition der einzelnen auszuführenden Aktivitäten sowie

eine Prozessausführungsumgebung vonnöten, um die Aktivitäten in den Prozessen zu interpretieren und auszuführen. Anforderungstechnisch sollte die Beschreibungssprache mächtig genug sein, um beliebige Aktivitäten und deren Kontrollfluss auf Basis der notwendigen Daten und Datentypen technologieunabhängig zu beschreiben und zu Geschäftsprozessen zusammenzusetzen sowie die Definition von Routinen zu ermöglichen, die Ausnahmezustände und Fehler bei der Prozessausführung behandeln. Gleichzeitig sollte eine Prozessbeschreibung aufgrund der Beschränkungen mobiler Endgeräte möglichst leistungs-, speicher- und damit auch energiesparsam ausführbar sein. Damit die Benutzerinteressen gewahrt werden, sollte ferner die Möglichkeit bestehen, Qualitätsaspekte - z. B. über die Spezifikation von nicht funktionalen Aspekten bei der Prozessdefinition - festzulegen, mit deren Hilfe die Ausführung und Migration nach den Wünschen des Benutzers gesteuert wird. In Hinblick auf die Prozessausführung ist zu berücksichtigen, dass die jeweiligen Geräte sich potenziell stark in ihren Attributen wie Leistung, Akkulaufzeit etc. unterscheiden, weshalb die Umgebung möglichst unterschiedliche Stufen an Performanz und Leistungen unterstützen sollte, um ein hohes Kooperationspotenzial zu erreichen. [KZL06]

Da gerade in mobilen Umgebungen aufgrund der Mobilität a priori nicht vorhergesagt werden kann, welche Endgeräte und welche Dienste zum Ausführungszeitpunkt eines Prozesses zur Verfügung stehen, ist darüber hinaus die Möglichkeit der Bindung zur Laufzeit an die jeweiligen Dienste essenziell. Auf diese Weise lässt sich eine den Prinzipien der serviceorientierten Architektur angemessene Flexibilität in Form einer losen Kopplung zwischen den Endgeräten erreichen (vgl. 2.3.1). Späte Bindung und Migration können allerdings nur dann stattfinden, wenn die einzelnen Endgeräte bzw. die zugehörige Middleware Funktionalität bereitstellt, mit deren Hilfe die Nachbarschaft permanent oder periodisch sondiert wird, um festzustellen, welche Services und Endgeräte zu einem Zeitpunkt zur Verfügung stehen. Diese Kontextinformationen müssen also akquiriert und verarbeitet werden, d.h., die Middleware muss context-aware (vgl. 2.3.2) sein, um sich den spezifischen Änderungen anzupassen. [KZL07a], [KZL06]

Beispiel

Um das Kooperationskonzept der mobilen Prozesse zu verdeutlichen, wird im Folgenden ein abstraktes Beispiel vorgestellt:

Abbildung 2.6 beinhaltet drei heterogene Endgeräte (PDA, Mobiltelefon, Notebook), die auf Basis von mobilen Prozessen kooperieren, um einen Geschäftsprozess gemeinsam zu bearbeiten. Im ersten Schritt definiert der Besitzer des PDAs mithilfe einer Spezifikationssprache einen Prozess, der sich aus drei Aktivitäten zusammensetzt und initiiert dessen Ausführung. Solange der PDA die Fähigkeit besitzt, den Prozess bzw. die als Nächstes anstehende Aktivität zu bearbeiten, ist keine Migration vonnöten. In diesem Beispiel führt der PDA die erste der drei Aktivitäten aus, hat aber nicht die entsprechenden Fähigkeiten bzw. Ressourcen, um die zweite Aktivität auszuführen. Damit

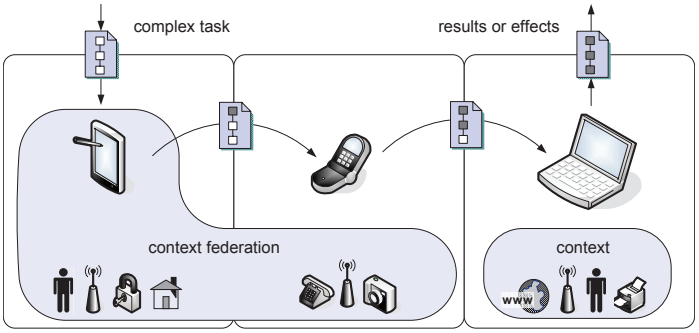

Abbildung 2.6: Kontextbasierte Kooperation über mobile Prozesse (aus [KZTL08])

der Prozess trotzdem erfolgreich abgeschlossen werden kann, ist Migration erforderlich, d.h., der PDA sucht auf Basis der jeweiligen Kontextinformationen ein Endgerät, das die zweite anstehende Aktivität bearbeiten kann. Im Beispiel findet der PDA ein Mobiltelefon und transferiert den kompletten Prozess inklusive der zugehörigen Statusinformationen (u.a. Ausführungszustand) auf das Mobiltelefon, das im Anschluss die Arbeit an der zuvor unterbrochenen Stelle fortsetzt. Für die dritte Aktivität ist das Mobiltelefon wiederum nicht geeignet, sodass eine erneute Migration in Schritt drei zu einem Notebook durchgeführt wird, das die dritte und letzte Aktivität und damit die Prozessbearbeitung abschließt. Je nach Definition des Prozesses kann im Anschluss das Ergebnis der Ausführung an den Prozessinitiator weitergeleitet werden.

Projekt Distributed Environment for Mobility-Aware Computing

Im Rahmen des Forschungsprojektes *Distributed Environment for Mobility-Aware Computing* (kurz DEMAC) wurde an der Universität Hamburg eine serviceorientierte kontextbezogene Middleware zur Unterstützung von mobilen Prozessen in mobilen Umgebungen entwickelt, die die bereits genannten Anforderungen an eine adäquate Infrastruktur berücksichtigt. Im Folgenden wird die zugehörige Middleware-Architektur vorgestellt. Abbildung 2.7 gibt einen grafischen Überblick über die einzelnen Komponenten der Architektur.

- **Kommunikation:** Für die Kommunikation mit anderen in der Umgebung befindlichen Endgeräten ist die Kommunikationskomponente, welche sich aus dem *Asynchronous Transport Service* sowie dem *Event Service* zusammensetzt, zuständig. Während der *Asynchronous Transport Service* Basisfunktionalität wie das asynchrone Senden und Empfangen von Nachrichten ermöglicht, stellt der *Event Service* Funktionalität bereit, die über Events (spezielle Nachrichten) registrierte Interessenten auf eigene Zustandsänderungen sowie Zustandsänderungen in der Umgebung hinweist. Da sich die Kommunikationsmöglichkeiten von Endgeräten zu Endgerät häufig stark unterscheiden, wird von konkreten Transportprotokollen wie Blue-

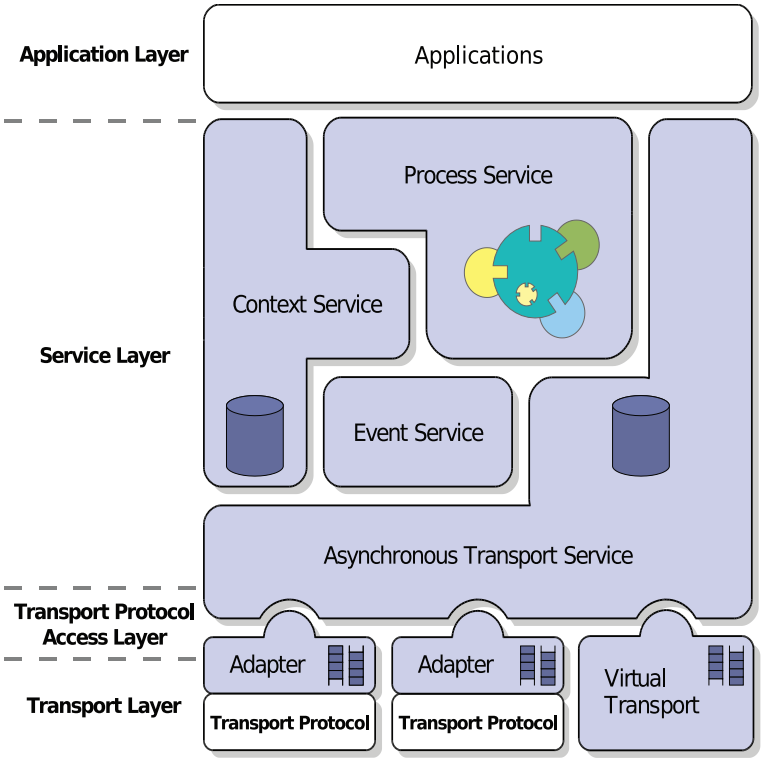

Abbildung 2.7: Abstrakte DEMAC Architektur (aus [KZL07b])

tooth, IrDa oder TCP/IP abstrahiert und ein eigenes Adressierungsschema verwendet. Sofern Endgeräte über mehrere konkrete Protokolle verfügen, können darüber hinaus Qualitätsaspekte bei der Wahl berücksichtigt werden. [KZL07b] [Kun05]

- **Kontextverwaltung:** Der *Context Service* hat die Aufgabe, sämtliche Kontextinformationen eines Endgerätes zu sammeln und zu verwalten. Die Akquisition von Kontextinformationen läuft hierbei entweder über Eventnachrichten über den direkten Nachrichtenaustausch oder über Sensoren ab, wobei ein föderierter Ansatz angestrebt wird, d.h., jedes Endgerät stellt lediglich lokale Kontextinformation bereit. Der globale Kontext kann hierbei durch Vereinigung sämtlicher lokaler Kontextinformationen erlangt werden. [KZL07b]

Definition und Ausführung von Prozessen: Der *Process Service* erlaubt die Definition, Integration und Ausführung von mobilen Prozessen. Er besteht aus zwei Hauptkomponenten:

- **Prozessbeschreibungssprache:** Aufgrund der spezifischen Eigenschaften von dynamischen mobilen Systemen und ihrer Knoten, wie sie zum Beispiel in MANETs vorzufinden sind, muss eine geeignete Prozessbeschreibungssprache die Definition von flexiblen und in hohem Maße fehlertoleranten mobilen Prozessen ermöglichen (vgl. 2.3.3). Im Projekt DEMAC ist hierfür die Sprache *DEMAC Process Description*

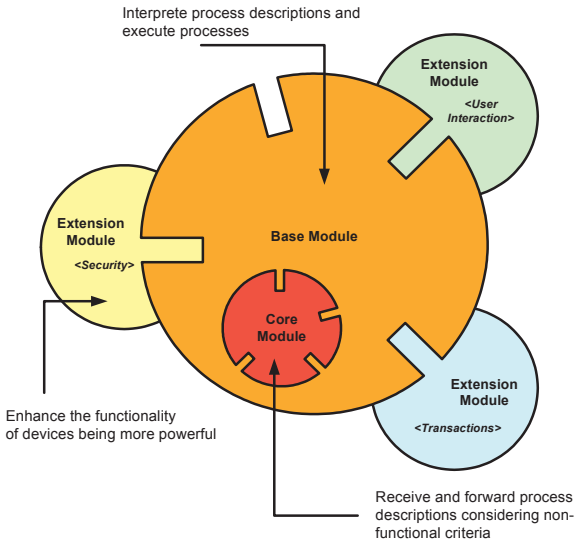

Abbildung 2.8: Modulare Ausführungsumgebung für mobile Prozesse (aus [KZL06])

Language (DPDL) vorgesehen, die XML-basiert und speziell auf die Definition von mobilen Prozessen zugeschnitten ist. [ZK07]

- **Prozessausführungsumgebung:** Die Prozessausführungsumgebung erlaubt die Interpretation und Ausführung der Aktivitäten eines Prozesses. Sofern ein Prozess lokal nicht weiter bearbeitet werden kann, ermöglicht die Umgebung darüber hinaus auf Basis der durch den Context Service bereitgestellten Kontextinformationen den gezielten Transfer eines Prozesses zu anderen potenten Geräten. Im Hinblick auf die Migration wird hierbei der Ansatz der starken senderinitiierten Codemobilität unterstützt, wobei echte Migration und nicht das Klonen von Prozessen stattfindet (vgl. 2.2.2). Aufgrund der Diversität von unterschiedlichen Endgeräten in Bezug auf ihre Leistungsparameter und Ausstattung unterstützt die Ausführungsumgebung unterschiedliche Performanzlevel. So müssen leistungsarme Endgeräte lediglich die Kernfunktionalität, Prozesse auf ein anderes Gerät zu transferieren, besitzen, während für leistungsstarke Geräte die Möglichkeit besteht, diese Kernfunktionalität um ein Basismodul, das die Interpretation und Ausführung von Prozessen ermöglicht sowie über Plug-Ins (Extension Module) um Aspekte wie Sicherheit, Transaktionen etc. zu erweitern. [KZL06] Abbildung 2.8 verdeutlicht das verwendete Konzept der Prozessausführungsumgebung grafisch.

2.4 Zusammenfassung

Das vorliegende Kapitel hat gezeigt, wodurch sich verteilte Systeme auszeichnen und anhand ihrer Vorteile herausgestellt, warum vermehrt verteilte Ansätze bei der Entwick-

lung von Systemen gewählt werden. Basierend auf diesem Wissen wurden die mobilen Systeme eingeführt, die aufgrund der vielen Möglichkeiten, die die Mobilität von Endgeräten birgt, heutzutage eine wichtige Rolle spielen. Hierbei wurde vermittelt, dass mobile Endgeräte gerade aufgrund ihrer per Definition möglichen Mobilität und den damit verbundenen Vorteilen leider auch mannigfaltigen Beschränkungen unterliegen, die bei der Entwicklung von Infrastrukturen für mobile Systeme berücksichtigt werden müssen (vgl. 2.2.4).

Infrastrukturlose spontane Netzvarianten wie die hier vorgestellten mobilen Ad-hoc-Netze (vgl. 2.2.7) stellen eine weitere Herausforderung in mobilen Systemen dar, da sie zwar leicht, kostenarm und auch in Gegenden, in denen keine Infrastruktur aufgebaut werden kann, möglich sind, aber einen höheren Kooperationsgrad der einzelnen Teilnehmer vor allem im Bereich des Routings, der Adressierung und in der allgemeinen Verwaltung benötigen.

Als besonderes Kooperationskonzept wurden die mobilen Prozesse vorgestellt (vgl. 2.3.3), die hochgradig flexible Kooperation in stark volatilen Umgebungen ermöglichen, indem kontextbezogene starke Migration (vgl. 2.2.2) von (Geschäfts-)Prozessen ermöglicht wird, um damit das Potenzial leistungsfähigerer benachbarter Endgeräte zu nutzen.

Die Kooperation führt aber aufgrund der Mobilität und der hiermit einhergehenden hohen Fehlerwahrscheinlichkeit nur dann zu einer adäquaten Kooperationsgüte, wenn angepasste Managementmechanismen integriert werden, die die Volatilität und ihre Implikationen berücksichtigen. Neben automatisierten Recoverymaßnahmen sind hiermit auch Logging- und Monitoring-Mechanismen gemeint, mit denen die Prozessausführung im System kontrolliert und überwacht werden kann (vgl. 1.2). Das folgende Kapitel beschäftigt sich daher mit dem Management mobiler Prozesse.

3 Ansätze zum Management mobiler Prozesse im Mobile Computing

3.1 Management

In diesem Abschnitt wird zunächst allgemein erläutert, was genau unter Management verstanden wird. Im Anschluss wird das Workflow-Management vorgestellt, da diese in Unternehmen oft verwendete Art von Management im Hinblick auf einige Charakteristika Ähnlichkeit mit der Aufgabe des Managements von mobilen Prozessen hat und sich daher ggf. Managementfunktionen bzw. Managementarten ermitteln lassen, die auch in mobilen Systemen relevant sind.

3.1.1 Definition und Abgrenzung

Der Begriff *Management* kommt ursprünglich aus dem anglo-amerikanischen Raum, hat allerdings heutzutage bereits den Einzug in die deutsche Sprache gefunden. Die Wortherkunft des zu Grunde liegenden Verbes *to manage* konnte bis heute nicht einwandfrei geklärt werden. Eine plausible Interpretation bezüglich der Etymologie des Verbes besagt zum Beispiel, wonach das Wort vom lateinischen „manus agere“, was in etwa „an der Hand führen“ bedeutet, abstammt, wobei auch die Interpretation, dass sich das Wort vom italienischen „maneggiare“ („handhaben, bewerkstelligen“) ableitet, durchaus plausibel erscheint. [BLP04] [Sim08] Im anglo-amerikanischen Raum existieren typischerweise zwei Bedeutungsvarianten des Begriffs *Management*:

- **institutional:** Im *institutionalen* Fall steht der Begriff Management für eine Personengruppe, die ein Unternehmen führt (z. B. Unternehmer, Vorstände, Geschäftsführer). Dies sind folglich diejenigen Personen, die eine Entscheidungs- und Anordnungsgewalt gegenüber den Personen auf der operativen Ebene (Mitarbeitern) haben. [Sim08]

- **funktional:** Im *funktionalen* Sinne bezeichnet der Begriff Management eine Abfolge von spezifischen Funktionen. Eine einheitliche Bezeichnung, welche Funktionen der funktionale Managementbegriff umfasst, existiert nicht. Viele Abhandlungen beziehen sich auf die von *Henry Fayol* (1841-1925) erstellte Einteilung, in der die Funktionen *Vorschau und Planung* (prévoir), *Organisation* (organiser), *Leitung* (commander), *Koordination* (coordonner) und *Kontrolle* (controller) den Begriff umschreiben. [BLP04]

Wie die oben aufgezeigte Begriffserklärung verdeutlicht, hat das Wort „Management“ einen stark betriebswirtschaftlich geprägten Charakter, weshalb es nicht verwunderlich

ist, dass gerade in dieser Disziplin mannigfaltige Managementvarianten (u.a. Customer-, Relationship-Management, Supply-Chain-Management) existieren. Neben der oben aufgezeigten allgemeinen Definition existieren aber auch Varianten für das Management in IT-Systemen:

- *Management* (bezogen auf IT-Infrastrukturen) umfasst nach *Hegering* die Gesamtheit sämtlicher Aktivitäten und Vorkehrungen zur Sicherstellung eines effektiven Einsatzes des Systems, seiner Dienste und Anwendungen. Dies bezieht sich sowohl auf technische Komponenten, Programme und Verfahren als auch auf Personal und Geschäftsprozesse. [Heg05]

Die spezifisch auf Computersysteme ausgerichtete Definition verdeutlicht bereits relativ gut, was das Ziel des Managements in IT-Systemen ist. Der Satzteil „[...]sämtliche Aktivitäten und Vorkehrungen zur Sicherstellung eines effektiven Einsatzes des Systems[...]“ ist aber noch sehr unspezifisch, da diese Definition auf IT-Systeme im Allgemeinen ausgerichtet ist und es natürlich von den jeweiligen Charakteristika und Gegebenheiten des jeweiligen Systems abhängt, welche Aktivitäten und Vorkehrungen notwendig sind, um das genannte Ziel zu erreichen.

Um nun zu verdeutlichen, welche Funktionen bzw. Aktivitäten und Vorkehrungen das Management in mobilen Systemen im Detail umfasst, wird zunächst das häufig in Unternehmen verwendete Workflow-Management betrachtet, da bei dieser Art des Managements ebenfalls Aspekte wie Verteilung, Computerunterstützung, -automatisierung sowie (Geschäfts-)Prozesse existieren und es daher naheliegt, dass einige Gesichtspunkte und Managementfunktionen auch für mobile Prozesse relevant sind.

3.1.2 Workflow-Management in stationären Systemen

Bevor der Begriff des Workflow-Managements erklärt wird, werden zunächst die häufig synonym verwendeten Begriffe Geschäftsprozess und Workflow definiert sowie deren Unterschiede herausgestellt. Im Anschluss werden die Aufgaben und Ziele beim traditionellen Workflow-Management verdeutlicht.

Geschäftsprozess

Nach *Hammer und Champy* wird unter einem *Geschäftsprozess* eine Menge von Aktivitäten verstanden, für die ein oder mehrere verschiedene Eingaben benötigt werden und die für einen Kunden ein Ergebnis von Wert erzeugen. [HC94] Sie beschreiben folglich die zur Erstellung von Produkten und/oder Leistungen erforderlichen betrieblichen Abläufe. Um wirtschaftlichen Erfolg zu haben, ist es notwendig, dass sich Unternehmen intensiv mit der Gestaltung sowie dem Management von Geschäftsprozessen befassen, um heutige Herausforderungen wie kürzere Produktlebenszyklen, steigende Kundenanforderungen, gesetzliche Anforderungen und Normen, nationale und internationale Konkurrenz

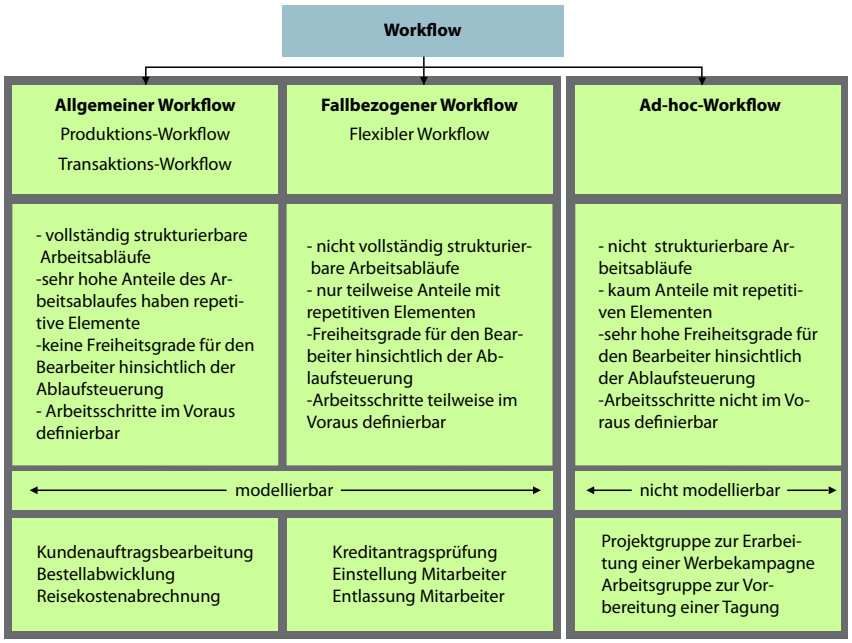

Abbildung 3.1: Workflows nach dem Strukturierungsgrad (nach [Gad08])

und den steigenden Kostendruck adäquat und effizient zu bewältigen und damit wettbewerbsfähig zu bleiben. Das *Geschäftsprozessmanagement* umfasst hierbei die systematische Gestaltung, Überwachung, Steuerung und Weiterentwicklung aller Geschäftsprozesse eines Unternehmens. [All07]

Workflow

Unter einem Workflow wird nach *Gadatsch* ein formal definierter, vollständig oder teilweise automatisierter Geschäftsprozess verstanden. Er enthält die zeitlichen, fachlichen und ressourcenbezogenen Spezifikationen, die für eine automatisierte Steuerung des jeweiligen Arbeitsablaufes auf der ausführenden (operativen) Ebene notwendig sind. Die einzelnen Aktivitäten innerhalb des beschriebenen Arbeitsablaufes sind hierbei zur Ausführung durch spezielle Anwendungsprogramme oder Mitarbeiter vorgesehen. Als *Workflow-Instanz* wird im Gegensatz zum Workflow als Schema oder Typ eines ganz bzw. teilweise automatisierten Arbeitsablaufes die konkrete Ausführung eines Workflows bezeichnet. [Gad03]

Workflows lassen sich anhand des Strukturierungsgrades (vgl. Abbildung 3.1) sowie des Grades der Computerunterstützung unterscheiden. [Gad03] *Allgemeine Workflows* (auch *Produktions-Workflows* oder *Transaktions-Workflows*) zeichnen sich dadurch aus, dass sie gut strukturierbare Arbeitsabläufe beschreiben und sich die einzelnen Aktivitäten bzw. Elemente der Arbeitsabläufe häufig wiederholen. Diese Art von Workflows ist aufgrund dieser Eigenschaften besonders gut im Voraus modellierbar und kann daher auch in hohem Maße von Computern unterstützt und ggf. sogar vollständig automatisiert

ausgeführt werden. *Fallbezogene Workflows* (auch *flexible Workflows*) lassen sich im Gegensatz zu *allgemeinen Workflows* nicht vollständig strukturieren und haben auch nicht den hohen Grad an sich wiederholenden Elementen innerhalb der Arbeitsabläufe. Dies wiederum führt zu mehr Freiheiten in der Steuerung des Ablaufes für den jeweiligen Bearbeiter, allerdings lassen sich diese Workflows aufgrund dieser Freiheiten und der genannten Eigenschaften nicht ganz so gut automatisieren und im Voraus modellieren. *Ad-hoc-Workflows* sind im Grunde genommen das Gegenteil von *allgemeinen Workflows*, d.h., die jeweiligen Arbeitsabläufe lassen sich gar nicht strukturieren und sie weisen auch nur geringe Anteile sich wiederholender Aktivitäten auf, was zu hohen Freiheitsgraden für den Bearbeiter führt, aber eine Modellierung verhindert. Charakterisiert man Workflows danach, wie gut sie von Computern unterstützt werden, so lassen sich drei Arten unterscheiden: *freie Workflows* werden komplett manuell von einer Person bearbeitet, während die jeweilige Person bei *teilautomatisierten Workflows* zum Beispiel von einem Informationsverarbeitungsprogramm unterstützt wird. Ein *automatisierter Workflow* wiederum läuft vollkommen automatisch ab, also ohne manuellen Eingriff ab.

Die für diese Arbeit interessantesten Workflows aus der ersten genannten Kategorie sind die *allgemeinen Workflows*, da sie a priori definiert werden und damit potenziell ohne weitere Benutzerinteraktion in Form von mobilen Prozessen in mobilen Ad-hoc-Netzen dezentral ausgeführt werden können, um die jeweiligen Ziele des Prozessinitiators zu erfüllen. Bezogen auf die letztgenannte Klassifizierung sind vor allem *automatisierte Workflows* relevant, da Benutzerinteraktionen generell erfordern, dass der jeweilige Prozessbearbeiter detailliertes Wissen über einen Workflow besitzt, was in potenziell hoch dynamischen Umgebungen, in denen a priori nicht vorhergesagt werden kann, welches Endgerät welche Aktivität eines Workflows bzw. Prozesses bearbeitet, meistens eher nicht gegeben ist.

Workflow-Managementsystem

Nachdem nun die Begriffe Geschäftsprozess sowie Workflow betrachtet wurden, kann auf Basis dieser Begriffe das Workflow-Management definiert werden. Festzuhalten bleibt allerdings, dass weder für den Begriff des Workflow-Managements (WFM) noch für den Begriff des Workflow-Managementsystems (WFMS) einheitliche Definitionen und damit auch kein einheitliches Verständnis, welche Funktionen diese Art des Managements umfasst, bestehen und daher jeweils eine für diese Arbeit geeignete Definition ausgewählt wurde. Nach *Gadatsch* bezeichnet der Begriff des *Workflow-Managements* Methoden und Werkzeuge zur computergestützten Planung, Simulation, Analyse, Steuerung und Überwachung von Arbeitsabläufen. Workflow-Management ist demnach ein operatives Konzept, mit dessen Hilfe die jeweiligen übergeordneten Unternehmensziele möglichst effizient umgesetzt werden sollen. [Gad03] Um aktiv Workflow-Management zu betreiben, werden zur Unterstützung Workflow-Managementsysteme verwendet, die im Folgenden näher betrachtet werden:

> *„Ein Workflow-Managementsystem ist ein anwendungsunabhängiges, dem Middlewarebereich zuzuordnendes Softwaresystem, das die Modellierung, die Ausführung, und das Monitoring von Workflows, sowie gegebenenfalls weitere Funktionen wie die Simulation und die Analyse von Workflows, unterstützt; insbesondere ist es in der Lage (semi-)formale Workflow-Spezifikationen zu interpretieren, die Ausführung von Prozessschritten durch die vorgesehenen Aktivitätsträger - Mitarbeiter oder Anwendungsprogramme - zu veranlassen und gegebenenfalls erforderliche Arbeitsanweisungen, Werkzeuge, Anwendungsprogramme, Informationen und Dokumente bereitzustellen."* [Gad03, S.210]

Um die Aspekte und Aufgaben eines Workflow-Managementsystems und damit auch des Workflow-Managements zu verdeutlichen, werden im Folgenden die einzelnen Hauptfunktionen näher betrachtet.

Bevor ein Workflow ausgeführt werden kann, muss der Schritt der Modellierung gegangen werden: *Modellieren* bedeutet in diesem Kontext, dass ein Modell von etwas hergestellt wird. Ein *Modell* wiederum ist definiert als eine Vorstellung, die sich eine Person oder allgemeiner ein Individuum von einem Gegenstand oder einem Vorgang in seiner Umwelt macht. [Jab97] Ziel der Modellierung ist es, die einzelnen Arbeitsabläufe eines Geschäftsprozesses formal zu beschreiben, also ein Workflow-Schema (vgl. 3.1.2) zu einem Geschäftsprozess zu erstellen. Bei der Modellierung wird der jeweilige Benutzer hierbei vom WFMS durch adäquate Werkzeuge unterstützt. Der Schritt *Ausführung* umfasst die konkrete Instanziierung und Ausführung eines Workflows. Die Aufgaben des WFMS bestehen hierbei darin, die Workflow-Definition zu interpretieren und die einzelnen Arbeitsschritte an die in der Definition vorgesehenen Mitarbeiter oder Anwendungsprogramme zu delegieren, wobei notwendige Informationen und Parameter ebenfalls mit übergeben werden.

Monitoring umfasst sowohl aktive als auch passive Überwachungsaufgaben. Zu den passiven zählen zum Beispiel Logging-Aufgaben wie die Ermittlung und Bereitstellung von Statusinformationen über die laufenden Vorgänge oder die Auslastung der jeweiligen Ressourcen. Aktive Überwachungsaufgaben betreffen zum Beispiel die Überwachung der Start- und Endtermine von Vorgängen mit dem Ziel, Probleme frühzeitig zu erkennen und damit eine Blockierung von Arbeitsabläufen zu verhindern. Sollte ein Mitarbeiter zum Beispiel krank werden, ist es sinnvoll, dass sämtliche Vorgänge, die dem kranken Mitarbeiter zugewiesen wurden, über Ausnahmeroutinen auf einen Stellvertreter umdelegiert werden. [Gad03]

3.2 Management mobiler Prozesse

Zunächst werden anhand eines Vergleichs zwischen dem im vorherigen Kapitel beschriebenen Workflow-Management und dem Management mobiler Prozesse wichtige Managementarten herausgestellt sowie deren Unterschiede und hieraus resultierende Proble-

me erläutert. Im Anschluss wird der derzeitige Forschungsstand des Managements mobiler Prozesse aufgezeigt, um zu verdeutlichen, welche Aspekte und Konzepte auf diesem Gebiet bereits adäquat umgesetzt wurden und für welche noch Lösungen benötigt werden. Auf Grundlage dieser Informationen werden die Probleme des Managements mobiler Prozesse im Detail verdeutlicht. Im letzten Abschnitt werden die gewonnenen Erkenntnisse zu einer Definition des Begriffs „Management mobiler Prozesse“ verdichtet.

3.2.1 Vergleich des Workflow-Managements mit dem Management mobiler Prozesse

Die Definition des Workflow-Managementsystems kennzeichnet Modellierung, Ausführung und Monitoring als die zentralen Funktionen, die ein derartiges System zu leisten hat. (vgl. 3.1.2) Das Workflow-Management wird in Unternehmen eingesetzt, um die Arbeitsabläufe zu optimieren, zu analysieren, zu überwachen und damit die Unternehmensziele effizient umzusetzen. Die einzelnen Arbeitsabläufe werden hierbei im Auftrag des Unternehmens vornehmlich in einer leistungsfähigen und zuverlässigen Umgebung, also zumeist auf High-End-Servern durchgeführt, die drahtgebunden miteinander vernetzt sind. Die Workflow-Kontrolle und Workflow-Steuerung wird in diesem Szenario zentralisiert gestaltet. [NFOP07] Des Weiteren ist im Voraus bekannt welche Mitarbeiter oder Programme innerhalb des Unternehmens welche Aktivitäten des jeweiligen Workflows erledigen können. Dies wird gemeinhin durch die Angabe einer fixen Instanz (z. B. „Mr. Smith“) oder einer Rolle (z. B. „Kreditsachbearbeiter“) gehandhabt, wobei das Letztgenannte zur Ausführungszeit einer fixen Instanz anhand von Kriterien wie z. B. der derzeitigen Auslastung zugewiesen werden kann. [GHS95]

Mobile Prozesse sind im Gegensatz zu den genannten Workflows oder betrieblichen Geschäftsprozessen benutzerzentriert, d.h., sie werden im Auftrag eines Benutzers auf dezentrale Art und Weise ausgeführt, wobei aufgrund der Migration auch die Kontrolle über den Prozess abgegeben wird. In einem System, das über mobile Prozesse kooperiert, kann folglich im Prinzip jeder Benutzer eigene Prozesse starten, während dies bei betrieblichen Geschäftsprozessen in Unternehmen in der Regel nicht der Fall ist. Speziell in MANETs (vgl. 2.2.7) kommt es im Gegensatz zu einem zuverlässigen Unternehmensnetzwerk nach oben beschriebenem Schema potenziell zu ständigen mobilitätsbedingten Topologieänderungen, außerdem sind die Kommunikationsbeziehungen zwischen den einzelnen Knoten aufgrund der hohen Fehleranfälligkeit der mobilen Endgeräte und der häufig verwendeten drahtlosen Kommunikation deutlich unzuverlässiger (vgl. 2.2.4). Die Mobilität führt des Weiteren dazu, dass keineswegs vorhergesagt werden kann, welche Aktivität eines Prozesses von welchem Benutzer bearbeitet wird, da nicht bekannt ist, welche Geräte oder Benutzer zu welchem Zeitpunkt verfügbar sind.

Die zentralen Funktionen eines Workflow-Managementsystems sind hierbei unter anderem auch in einem Szenario mit mobilen Prozessen notwendig: Die *Modellierung* über

eine Spezifikationssprache und *Ausführung* über eine Ausführungsumgebung sind Grundvoraussetzungen, damit mobile Prozesse verwendet werden können (vgl. 2.3.3). Das *Monitoring*, also die Überwachung, hat ebenfalls zentrale Bedeutung, ist aber schwerer umzusetzen, da aufgrund der Migrationsmöglichkeit der mobilen Prozesse ggf. die Kontrolle über den jeweiligen Prozess abgegeben wird und damit die Überwachung in einer unbeständigen Umgebung ohne zentrale unabhängige Komponente schwerer ist als bei traditionellen Workflows, bei denen eine zentralisierte Kontrollinstanz permanent den Überblick hat, wie weit die Bearbeitung eines Workflows fortgeschritten ist.

Weiterhin gilt, dass bei einer zentralen Steuerung relativ einfach, z. B. vom Server, der den Workflow steuert und die Aufgaben zuweist, festgehalten werden kann, welche Person oder welches Programm eine bestimmte Aktivität eines Geschäftsprozesses durchgeführt hat oder derzeit durchführt. Derartige *Logging*-Mechanismen sind im mobilen Szenario problematischer, da z. B. in MANETs eine zentrale und steuernde Instanz fehlt und auch keine feste Infrastruktur vorhanden ist, die feststellen kann, wer welche der Aktivitäten eines Prozesses durchgeführt hat und/oder wohin ein Prozess migriert ist.

Die erhöhte Fehleranfälligkeit und Mobilität in mobilen Systemen erfordert darüber hinaus angepasste *Recovery*-Mechanismen, da Fehler hier im Gegensatz zu einem zuverlässigen Unternehmensnetzwerk nicht die Ausnahme, sondern die Regel sind.

Modellierung, Ausführung, Monitoring-, Logging- und Recovery-Mechanismen sind zusammenfassend in einem mobilen System, das die Kooperation mithilfe von mobilen Prozessen unterstützt also essenzielle Managementkomponenten, die aber aufgrund der unterschiedlichen Voraussetzungen und Gegebenheiten nicht direkt aus dem traditionellen Workflow-Management übernommen werden können. Infolge der aus der Mobilität resultierenden mannigfaltigen Restriktionen ist die Umsetzung der genannten Funktionalität dementsprechend in der Regel schwerer.

3.2.2 Ist-Zustand des Managements mobiler Prozesse

Der vorangehende Vergleich zwischen dem Management mobiler Prozesse und dem Workflow-Management hat verdeutlicht, dass Modellierung, Ausführung, Monitoring-, Logging- und Recovery-Mechanismen wünschenswerte und teils sogar obligatorische Aspekte sind, um das Kooperationskonzept der mobilen Prozesse in volatilen Umgebungen adäquat und effizient umzusetzen.

Nach derzeitigem Forschungsstand existieren sowohl Konzepte für eine geeignete Spezifikationssprache als auch für eine Ausführungsumgebung mobiler Prozesse. Ein Beispiel hierfür ist die XML-basierte Spezifikationssprache *DEMAC Process Description Language* (DPDL), die speziell auf die Spezifikation und Verwendung von mobilen Prozessen zugeschnitten ist und im Forschungsprojekt DEMAC verwendet wird (vgl. 2.3.3 bzw. [ZK07]) sowie die modulare Ausführungskomponente desselben Forschungsprojektes. Für beide Aspekte ist folglich keine Konzeptentwicklung notwendig. Für die restlichen drei genannten Managementaspekte (Monitoring, Logging, Recovery) existieren derzeit

keine zufrieden stellenden Konzepte, um die dahinterliegende Funktionalität in potenziell hoch dynamischen, ggf. infrastrukturlosen, fehleranfälligen Umgebungen umzusetzen. Da diese drei Funktionen aber essenziell sind, um eine adäquate und effiziente Kooperationsgüte zu erreichen, steht insbesondere die Entwicklung geeigneter Konzepte hierfür im Zentrum dieser Arbeit (vgl. 1.2).

3.2.3 Offene Probleme

In diesem Abschnitt werden die aus der Mobilität der Teilnehmer sowie die aus der Migrationsmöglichkeit der Prozesse entstehenden Probleme näher betrachtet, um zu verdeutlichen, welche Aspekte ein Managementsystem für mobile Prozesse im Detail behandeln sollte und welche Funktionen nützlich sind. Es wird hierbei erneut, sofern notwendig und sinnvoll, auf das in Kapitel 2.3.3 dargestellte abstrakte Beispiel zurückgegriffen, um die Probleme zu veranschaulichen. Sofern also Begriffe wie PDA, Mobiltelefon oder Notebook verwendet werden, bezieht sich dies auf das genannte Beispiel. Abbildung 2.6 verdeutlicht das zugehörige allgemeine Szenario grafisch.

- **Aus der Migration resultierende Probleme:** Sobald Prozess-Migration - beispielsweise wie im dargestellten Beispiel vom PDA zum Mobiltelefon -, stattfindet gibt der Sender die Kontrolle über seinen Prozess vollständig an den Empfänger ab. Da keine zentrale oder dezentrale Komponente existiert, die sicherstellt, dass der Prozess weiterhin ordnungsgemäß ausgeführt wird oder dass Statusinformationen aufgezeichnet werden, führt die Migration dementsprechend zur vollständigen Informations- und Kontrollarmut für den Prozessinitiator. Demnach ist ihm nicht bekannt, auf welchem Gerät sich der Prozess derzeit befindet, wie weit die Bearbeitung fortgeschritten ist und ob Probleme aufgetreten sind. Aufgrund der Beschränkungen und Charakteristika von mobilen Systemen ist es nicht unwahrscheinlich, dass Fehler auftreten, die eine erfolgreiche Prozessausführung gefährden. Fallen beispielsweise das Notebook oder das Mobiltelefon aus, während das jeweilige Gerät die Kontrolle über den Prozess hat, so ist dieser, sofern das Gerät nicht wieder auftaucht, verloren, ohne dass der Initiator oder ein anderes Gerät den Fehler bemerkt und sinnvolle Gegenmaßnahmen einleitet bzw. einleiten kann. Um derartige Prozessabbrüche zu vermeiden, werden Monitoring-Methoden zur Prozessüberwachung benötigt, damit Fehler überhaupt erkannt werden können und der Initiator möglichst jederzeit einen Überblick darüber hat, welches Gerät derzeit den Prozess kontrolliert und wie sein Bearbeitungsstatus ist. Darüber hinaus sind aber auch effiziente Recovery-Algorithmen essenziell, welche die erkannten Fehler, beispielsweise durch erneutes Wiederaufsetzen eines Prozesses, kompensieren. Diese Mechanismen sollten möglichst auch dann funktionieren, wenn das initiierende Gerät selbst, beispielsweise aufgrund von Energiearmut, (kurzzeitig) ausfällt oder anderen Fehlern ausgesetzt ist.

- **Problem der Feststellung der einzelnen Prozessbearbeiter:** Ein weiteres Problem besteht darin, dass nach (erfolgreichem) Abschluss oder während der Bearbeitung eines Prozesses nicht festgestellt werden kann, welche Geräte in welchem Umfang an der Bearbeitung eines Prozesses beteiligt waren. Derartige Logging-Funktionalität ist vor allem für Abrechnungszwecke wünschenswert, also zum Beispiel dann, wenn die jeweiligen Helfer für ihre Kooperationsbereitschaft entlohnt werden sollen.

- **Problem der Ergebnisrückgabe:** In vielen Fällen dürfte es für den Prozessinitiator wünschenswert sein zu erfahren, welche Ergebnisse der jeweilige Prozess produziert hat. Um diese Funktionalität zu erfüllen, ist folglich ein geeigneter Rückgabemechanismus notwendig, mit dessen Hilfe das Ergebnis bzw. die Ergebnisse vom letzten Glied der Prozessausführungskette zum ursprünglichen Initiator übermittelt werden können.

3.2.4 Definition und Abgrenzung

Nachdem das vorherige Kapitel aufgezeigt hat, welche Managementmethoden für Systeme, die mittels mobiler Prozesse kooperieren, relevant sind, wird der Begriff des Managements in diesem Kapitel definiert. Die folgende Definition bezieht sich folglich auf das Management mobiler Prozesse, wobei der Begriff diejenigen Managementarten umfasst, die für diese Arbeit als relevant identifiziert wurden.

Management mobiler Prozesse

Unter dem Begriff „Management mobiler Prozesse“ werden Logging-, Monitoring- sowie Recovery-Mechanismen verstanden. *Monitoring* bezeichnet die Überwachung der Prozessausführung: Es umfasst hierbei Funktionalität wie die Feststellung, auf welchem Endgerät sich ein laufender Prozess derzeit befindet, wie weit die Prozessbearbeitung abgeschlossen ist und ob Fehler aufgetreten sind. *Recovery* umfasst sämtliche Aktivitäten und Vorkehrungen, die es ermöglichen Fehler bei der Prozessausführung zu beheben. Unter *Logging* wird in diesem Kontext die Speicherung von ausführungsspezifischen Informationen verstanden, die z. B. für nachträgliche Analysen, zur Prozessoptimierung oder allgemein zur Verbesserung der Kooperationsqualität verwendet werden können. Hierzu zählt beispielsweise die Aufzeichnung der Identifikatoren der einzelnen Prozessbearbeiter zu Abrechnungszwecken.

Festzuhalten bleibt, dass sich die einzelnen Teilbereiche des Managements mobiler Prozesse (Logging, Monitoring und Recovery) durchaus in einigen Aspekten und Bereichen überlappen. Recovery benötigt zum Beispiel häufig Logdateien zur Fehlerbehandlung, ebenso sind derartige Protokolldateien auch für das Monitoring relevant. Um Recovery zu betreiben, müssen zunächst Fehler erkannt werden, demnach ist das Monitoring häufig auch eine wichtige Voraussetzung, um effektiv Recovery zu betreiben.

3.3 Anforderungen an eine Managementinfrastruktur für mobile Prozesse

In diesem Abschnitt werden die Anforderungen an eine Managementinfrastruktur für mobile Prozesse beschrieben. Neben allgemeinen Anforderungen werden spezifische, aus den Beschränkungen mobiler Systeme resultierende Anforderungen dargestellt sowie funktionale Aspekte behandelt.

3.3.1 Funktionale Anforderungen

- **Monitoring:** Eine Managementinfrastruktur sollte es generell ermöglichen, die Prozessausführung zu steuern und zu überwachen. Im Rahmen der Überwachung wird hierunter die autorisierte Feststellung, welches Gerät den jeweiligen Prozess derzeit bearbeitet, wie weit die Prozessbearbeitung abgeschlossen ist sowie die Feststellung von Fehlern bei der Prozessausführung verstanden, während die Steuerung das aktive Eingreifen in die eigentliche Bearbeitung beinhaltet.

- **Logging:** Die Managementkomponente sollte die Aufzeichnung und möglichst persistente Speicherung von beliebigen vom Prozesserzeuger festlegbaren, ausführungsspezifischen Informationen ermöglichen.

- **Recovery:** Ein Managementsystem sollte die Prozessausführung derart unterstützen und absichern, dass möglichst jede Art von Fehler kompensiert werden kann und damit insgesamt im System eine höhere Wahrscheinlichkeit der erfolgreichen Prozessausführung erreicht wird.

3.3.2 Allgemeine Anforderungen

- **Fehlertoleranz:** Eine Managementinfrastruktur sollte möglichst robust gegenüber Ausfällen von Knoten, Verbindungsabbrüchen oder anderen Fehlern sein und dennoch korrekt funktionieren.

- **Skalierbarkeit:** Ein Managementsystem sollte sowohl bei sinkender Anzahl als auch bei steigender Anzahl an Geräten bzw. Nutzern des Managementsystems im Netzwerk noch effizient funktionieren. (vgl. 2.1.3)

- **Verfügbarkeit:** Eine hohe *Verfügbarkeit* des Managementsystems ist wünschenswert. Unter *Verfügbarkeit* wird in diesem Zusammenhang die Wahrscheinlichkeit bezeichnet, dass ein System zu einem bestimmten Zeitpunkt korrekt arbeitet und damit seine Funktionalität den jeweiligen Interessenten zur Verfügung stellen kann. Hohe Verfügbarkeit bedeutet demnach, dass die Wahrscheinlichkeit sehr hoch ist, dass ein System zu jedem beliebigen Zeitpunkt korrekt funktioniert. [TS08]

- **Zuverlässigkeit:** Hierunter wird die Eigenschaft eines Systems verstanden, ausfallfrei zu laufen. *Zuverlässigkeit* bezieht sich hierbei auf ein Zeitintervall und nicht wie zum Beispiel die Verfügbarkeit auf einen Zeitpunkt. Hohe Zuverlässigkeit bedeutet wiederum, dass ein System mit hoher Wahrscheinlichkeit nicht ausfällt. [TS08] Dementsprechend ist für ein Managementsystem ein hoher Grad an Zuverlässigkeit sinnvoll und erstrebenswert.

- **Funktionssicherheit:** Hohe *Funktionssicherheit* bedeutet, dass ein vorübergehend nicht korrekt laufendes System keine „schlimmen" Folgen hat. [TS08] In diesem Kontext ist damit gemeint, dass ein (kurzzeitig) fehlerhaft arbeitendes Managementsystem die Prozessausführung nicht verschlechtert, d.h., dass die Wahrscheinlichkeit einer erfolgreichen Ausführung mit Managementsystem möglichst niemals kleiner ist als die Wahrscheinlichkeit einer Ausführung ohne ein Managementsystem. Dies impliziert, dass das Managementsystem generelle Funktionalitäten wie z. B. die Migration, die lokale Prozessausführung oder die Nutzung von Kontextinformationen nicht beeinträchtigt.

- **Effizienz:** Eine Infrastruktur zum Management sollte in hohem Maße effizient arbeiten, also hohen Nutzen bei möglichst geringem Aufwand bieten.

- **Erweiterbarkeit:** Eine Managementinfrastruktur sollte möglichst aufwandsarm um weitere Funktionalität bzw. Managementarten erweiterbar sein.

- **Nebenläufige Nutzung:** Es sollte parallel (bzw. nebenläufig) mehreren Benutzern bzw. Endgeräten möglich sein, das Managementsystem und seine Funktionalität in Anspruch zu nehmen, um inakzeptable Wartezeiten zu vermeiden und allgemein eine höhere Qualität des Managementsystems zu erreichen.

- **Sicherheit:** Gewisse Managementfunktionen wie zum Beispiel die Feststellung, wo sich ein Prozess zu einem Zeitpunkt befindet, sollten derart geschützt werden, dass lediglich befugte Endgeräte, also z. B. nur der Prozessinitiator, Zugriff auf diese Informationen hat. Um eine derartige Anforderung zu erfüllen, sind also unter anderem Authentisierungs- und Authentifizierungs-Mechanismen erforderlich.

- **Interoperabilität/Portabilität:** Die Managementkomponente sollte möglichst so spezifiziert werden, dass sie einfach und schnell ohne große Änderungen mit einer anderen Softwarekomponente/Middleware, für die sie ursprünglich nicht entwickelt wurde, zusammenarbeiten kann, um diese durch Managementfunktionalität zu erweitern.

- **Offenheit:** Um einheitlichen Zugriff auf das Managementsystem zu ermöglichen, sind standardisierte Schnittstellen notwendig, dies fördert wiederum die Portabilität und Interoperabilität. (vgl. 2.1.3)

- **Administration:** Der Administrationsaufwand einer Managementkomponente sollte sehr gering sein und möglichst wenig manuelles Eingreifen eines Benutzers erfordern. Hiermit sind sowohl der Aufbau als auch die Wartung und Rekonfiguration des jeweiligen Systems gemeint.

3.3.3 Aus den Restriktionen mobiler Systeme resultierende Anforderungen

- **Ressourcensparende Nutzung des Managementsystems:** Da mobile Endgeräte geringe Kapazitäten im Hinblick auf Geschwindigkeit, Speichergröße, Netzwerkbandbreite und Energiereserven haben (vgl. 2.2.4), sollte der Zugriff auf das Managementsystem möglichst ressourcenarm möglich sein, d.h., Anzahl und Größe der Nachrichten zwischen einem Knoten und dem Managementsystem sollten möglichst klein sein, da das Senden von Nachrichten sehr energieintensiv ist. [CDK05]

- **Leichtgewichtiges Managementsystem:** Neben dem Zugriff auf das Managementsystem sollte das Managementsystem selbst bezüglich des Speicherplatzes, des Rechenaufwandes und auch des Koordinationsaufwandes möglichst sparsam sein sowie sich einfach, schnell und ressourcenarm aufbauen und in Fehlerfällen - wie zum Beispiel Knotenausfällen innerhalb des Systems - rekonfigurieren können. Diese Aspekte sind vor allem dann wichtig, wenn mobile Endgeräte als Einheiten Teil des Managementsystems sind.

- **Vielseitig einsetzbar:** Die Managementinfrastruktur sollte umgebungsunabhängig einsetzbar sein. Nach Möglichkeit sollte es folglich egal sein, ob ein System stationär, drahtgebunden und infrastrukturbasiert ist oder in hohem Grade mobil, drahtlos und infrastrukturlos, wobei natürlich auch beliebige Kombinationen aus beiden genannten Systemen gemeint sind.

- **Positionsunabhängigkeit:** Der Zugriff auf das Managementsystem sollte in hohem Maße unabhängig von der jeweiligen physischen Position eines (mobilen) Endgerätes sein, außerdem sollte ein Zugriff auch während der Bewegung eines Gerätes möglich sein.

- **Plattformunabhängigkeit:** Ein Managementsystem sollte in hohem Maße plattformunabhängig sein, damit potenziell jede Art von Endgerät unabhängig von der jeweiligen Architektur, dem Betriebssystem oder ähnlichen Charakteristika als Komponente eines solchen Systems auftreten oder es nutzen kann.

3.4 Betrachtung und Untersuchung bestehender Ansätze

In diesem Abschnitt werden konkrete bestehende Ansätze in Hinblick auf die zu realisierende Managementfunktionalität und die Charakteristika mobiler Systeme untersucht, um ggf. sinnvolle Methoden und Ideen aus anderen Bereichen zu adaptieren.

3.4.1 Recovery-Ansätze

Im Abschnitt 3.2 wurde bereits verdeutlicht, dass Recovery, also die Wiederherstellung eines Systems aus einem fehlerhaften in einen fehlerfreien Zustand, ein wichtiger Bestandteil der zu entwickelnden Managementkomponente ist, um trotz der per Definition vorhandenen Mobilität und der vorliegenden Restriktionen in mobilen Systemen eine hohe Fehlertoleranz zu erreichen. Als Nächstes werden daher gängige Recovery-Mechanismen und Ansätze für verteilte Systeme untersucht und es wird erörtert, inwiefern derartige Methoden auch in mobilen Umgebungen im Kontext von mobilen Prozessen einsetzbar sind.

Recovery in verteilten Systemen

Ziel der Recovery ist es, Fehler im System zu beheben, um damit seine Operabilität zu erhalten. Unter einem Fehler ist hierbei ein Systemzustand gemeint, der zu einem Ausfall führen kann. [TS08] Im Bereich der verteilten Systeme existieren zwei grundlegende Arten von Recovery-Mechanismen [TS08]:

- **Backward Recovery:** Wie der Name bereits zu erkennen gibt, geht es bei der *Rückwärtswiederherstellung* darum, das System von einem fehlerhaften Zustand in einen fehlerfreien älteren Zustand zurückzusetzen. Damit derartige Mechanismen möglich sind, ist Redundanz erforderlich, d.h., es müssen periodisch die Zustände des Systems (Kontrollpunkte) festgehalten werden, um im Falle eines Fehlers eine erfolgreiche Rücksetzung durchführen zu können. Das erneute Senden eines verloren gegangenen Pakets bei einer Interaktion zwischen zwei Kommunikationspartnern ist hierbei ein Beispiel für eine derartige Recovery.
- **Forward Recovery:** Bei der *Vorwärtswiederherstellung* wird keine Rücksetzung auf einen Kontrollpunkt durchgeführt, sondern es wird versucht, das System trotz des Fehlers in einen neuen fehlerfreien Zustand zu überführen. Ein Beispiel für eine derartige Recovery wäre das Wiederherstellen eines verloren gegangenen Pakets auf Basis später empfangener Pakete und einer Prüfsumme.

Im Gegensatz zur Backward Recovery ist es bei der Forward Recovery notwendig, dass a priori bekannt ist, welche Fehler auftreten können, um diese adäquat zu beheben und das System in einen neuen fehlerfreien Zustand zu überführen. Ein Nachteil der Rückwärtswiederherstellung ist hingegen, dass sich der Mechanismus des Rücksetzens negativ auf die Leistung des Systems auswirkt, da die Wiederherstellung eines alten Zustandes impliziert, dass gewisse Arbeitsschritte erneut wiederholt werden müssen und der Vorgang selbst sehr aufwändig sein kann. Des Weiteren kann bei diesem allgemeinen fehlerunabhängigen Mechanismus nicht garantiert werden, dass der jeweilige Fehler nicht erneut auftritt. Erschwerend kommt hinzu, dass in manchen Situationen und Systemen die Backward Recovery nicht durchführbar ist, da nicht jede durchgeführte

Aktion reversibel ist. [SS94] Die Zerstörung eines Autos auf einem Schrottplatz durch einen Roboter lässt sich beispielsweise normalerweise nicht mehr rückgängig machen.

Eine Vorwärtswiederherstellung erfordert, dass sämtliche Fehler im Voraus bekannt sind. Dies ist notwendig, damit das jeweilige System automatisch von einem fehlerhaften in einen fehlerfreien neuen Zustand überführt werden kann. Im Kontext der mobilen Prozesse ist diese Forderung nicht erfüllt, d.h., Vorwärtswiederherstellung ist hier nicht anwendbar. Im Folgenden wird daher die Idee der Rückwärtswiederherstellung näher betrachtet.

Backward Recovery im Kontext mobiler Prozesse

Damit die Wiederherstellung in einen älteren Zustand möglich ist, müssen die zugehörigen Kontrollpunkt-Informationen so gespeichert werden, dass sie Prozessabstürze, System- und Geräteausfälle überleben. In verteilten Systemen werden Daten hierzu häufig redundant auf unterschiedlichen, ggf. geografisch entfernten Festplatten gespeichert, um auch Festplattenausfälle und Katastrophen weitestgehend problemlos zu überstehen. [TS08]

Eine solche Vorgehensweise ist in mobilen Systemen in der Regel schwerer zu realisieren, da in einem mobilen Ad-hoc-Netzwerk beispielsweise a priori nicht vorhergesagt werden kann, welche Endgeräte zu welchem Zeitpunkt im Netz vorhanden oder erreichbar sind und eine Infrastruktur vorhanden ist, die die persistente Speicherung dieser Kontrollpunkte übernehmen kann. Hinzu kommt, dass Fehler und Ausfälle aufgrund der Restriktionen mobiler Endgeräte deutlich wahrscheinlicher als in stationären verteilten Systemen sind (vgl. 2.2.4). Die Speicherung der Kontrollpunkte (auch *Checkpoints* genannt) erfordert folglich je nach Art und Architektur des mobilen Systems angepasste Mechanismen, die trotz der Dynamik in diesen Systemen eine sichere Speicherung von Kontrollpunkten ermöglichen.

In vielen verteilten Systemen ist es notwendig, dass bei der Erstellung von Kontrollpunkten ein konsistenter globaler Zustand (auch *verteilte Momentaufnahme* oder *distributed snapshot*) aufgezeichnet wird. Basierend darauf existieren unterschiedliche Ansätze wie *unabhängige Kontrollpunkte* oder *koordinierte Kontrollpunkte* [TS08], die hier nicht näher behandelt werden, da die Erstellung einer verteilten Momentaufnahme des Systems speziell im Kontext der mobilen Prozesse nicht notwendig und sinnvoll ist. Mobile Prozesse sind mit Ausnahme von Subprozessen im Allgemeinen voneinander unabhängig, d.h., sie interagieren nicht untereinander, was impliziert, dass ein beispielsweise durch einen Geräteausfall hervorgerufener Fehler, der die Ausführung eines mobilen Prozesses auf einem Gerät beeinträchtigt, in der Regel keinen Einfluss auf mobile Prozesse auf anderen Geräten im Netzwerk hat, sodass lediglich der betroffene Prozess (und ggf. abhängige Subprozesse) wieder hergestellt werden muss. Demnach sind prozessspezifische Kontrollpunkte ausreichend und kein distributed snapshot notwendig. Für jeden mobilen

Prozess werden folglich anstatt eines systemglobalen Kontrollpunktes eigene unabhängige Kontrollpunkte benötigt.

Recovery in Workflow-Managementsystemen

Da im Bereich des Workflow-Managements ebenfalls Aspekte wie Verteilung, Computerautomatisierung und auch Prozesse eine wichtige Rolle spielen bzw. ein integraler Bestandteil sind, werden im Folgenden aufgrund dieser Analogien die Recovery-Mechanismen dieses Gebiets näher betrachtet und es wird weiterhin überprüft, ob derartige Methoden und Mechanismen auch für die Recovery mobiler Prozesse sinnvoll sind.

Überblick

Ziel der Recovery von Workflows ist es, nach einem Fehler, also beispielsweise wenn das jeweilige Workflow-Managementsystem (vgl. 3.1.2) aus einem beliebigen Grund ausfällt sämtliche, zum Zeitpunkt des Absturzes laufende Workflows, wiederherzustellen. Im Gegensatz zu Datenbanksystemen, in denen im Falle eines Absturzes nach dem Prinzip „alles oder nichts“ sämtliche Änderungen einer nicht erfolgreich durchgeführten Transaktion rückgängig gemacht werden, ist diese Art der Recovery bei Workflows nicht sinnvoll. Dies liegt daran, dass mittels Workflows Geschäftsprozesse (teil-)automatisiert werden, deren Aktivitäten zumeist langlebig (Stunden oder sogar Monate) sind, sodass eine komplette Neuaufsetzung des Prozesses zu einem inakzeptablen Arbeitsverlust sowie seine Wiederholung einen zu hohen Zeitaufwand impliziert, während in Datenbanksystemen die Transaktionen zumeist kurzlebige Aktivitäten beinhalten, die schnell und kostenarm erneut durchgeführt werden können. Abgesehen davon lassen sich bei Geschäftsprozessen Aktivitäten oftmals nicht erneut durchführen bzw. einfach rückgängig machen, sondern müssen semantisch korrigiert werden. So kann zum Beispiel eine per E-Mail durchgeführte Bestellung nicht einfach gestoppt werden, aber die Bestellung kann ggf. durch Senden einer weiteren E-Mail storniert und damit der Effekt kompensiert werden.

Um Inkonsistenzen zu vermeiden, aber trotzdem im Fehlerfall gerade bei langlebigen Prozessen nicht sämtliche Arbeitsschritte erneut durchführen zu müssen, werden daher bei Workflows spezielle *erweiterte Transaktions-Modelle* verwendet, bei denen die ACID-Eigenschaften (vgl. [LR00]) „aufgeweicht“ werden. Das am häufigsten verwendete *Erweiterte Transaktions-Modell* sind *offen verschachtelte Transaktionen* (*open nested transactions*). Eine *verschachtelte Transaktion* ist hierbei ein Baum von Sub-Transaktionen, wobei jede der Sub-Transaktionen wiederum eine beliebige Anzahl von weiteren Sub-Transaktionen haben kann. Nach Abschluss einer Sub-Transaktion werden die Änderungen sofort sichtbar, was kurze Sperrzeiten für Ressourcen impliziert und damit die Eignung gerade für langlebige Transaktionen (bzw. Workflows) erhöht. Damit trotz der schnellen Freigabe von Ressourcen Fehler-Atomarität gewahrt bleibt, sind so genannte *kompensierende Transak-*

tionen notwendig, mit deren Hilfe im Fehlerfall die Änderungen semantisch korrigiert werden können. [Mel07] [LR00] [GR93]

Workflow Recovery im Detail

Damit Workflows aufgrund von Fehlern nicht wieder ganz von vorne gestartet werden müssen, werden im Bereich der Workflow Recovery Kontextinformationen für jeden Workflow erhoben und persistent im Datenbankmanagementsystem gespeichert. Dieser Kontext beinhaltet unter anderem Informationen darüber, welche Aktivitäten innerhalb eines Workflows bereits abgeschlossen wurden und welche derzeit laufen. Damit die Kontextinformationen aktuell bleiben, müssen sie regelmäßig (z. B. nach Abschluss einer Aktivität) aktualisiert werden.

Im Fall eines verteilten Systems ist es notwendig, dass die jeweilige Komponente bzw. der jeweilige Knoten, der einen Workflow bzw. einzelne Aktivitäten eines Workflows bearbeitet, regelmäßig Informationen über den aktuellen Bearbeitungsstatus an das Datenbanksystem zum Zwecke der persistenten Speicherung übermittelt. Mithilfe dieser Informationen kann trotz eines System- bzw. Knotenausfalls jeder Workflow wiederhergestellt werden, sodass keine abgeschlossene Aktivität erneut durchgeführt werden muss und auch bekannt ist, welche Aktivitäten noch durchgeführt werden müssen. Für sämtliche zum Zeitpunkt eines Crashs laufende Aktivitäten müssen abhängig davon, ob die jeweilige Aktivität durch den Crash beeinflusst wurde, adäquate Aktionen durchgeführt werden. Für Aktivitäten, die nicht durch den Crash eines oder mehrerer Knoten im System beeinflusst wurden (also z. B. noch auf einem aktiven Knoten laufen und keine Abhängigkeit zu anderen ausgefallenen Aktivitäten besitzen), ist keine Recovery notwendig und die Bearbeitung kann direkt fortgesetzt werden. Sämtliche zum Zeitpunkt des Crashs aktiven Aktivitäten, die durch einen Crash beeinflusst werden, müssen abhängig davon, ob sie Teil einer Transaktion sind oder nicht Teil einer Transaktion sind, unterschiedlich behandelt werden.

Aktivitäten, die nicht im Kontext einer Transaktion ausgeführt wurden, können je nach Art der Aktivität und der bereits von ihr durchgeführten partiellen Änderungen direkt fortgesetzt, neu gestartet (wobei hierbei ggf. die partiellen Änderungen kompensiert werden müssen) oder anderweitig behandelt werden. Transaktionale Aktivitäten werden automatisch nach dem Neustart der jeweiligen Komponente abgebrochen und in einen konsistenten (älteren) Zustand zurückgesetzt, aus dem gefahrlos die Arbeit fortgesetzt werden kann, ohne die Konsistenz zu gefährden. [LR00] Abbildung 3.2 verdeutlicht die genannten Zusammenhänge und das Zusammenspiel der einzelnen Komponenten anhand der Momentaufnahme eines fiktiven Workflows.

Betrachtung im Kontext mobiler Prozesse

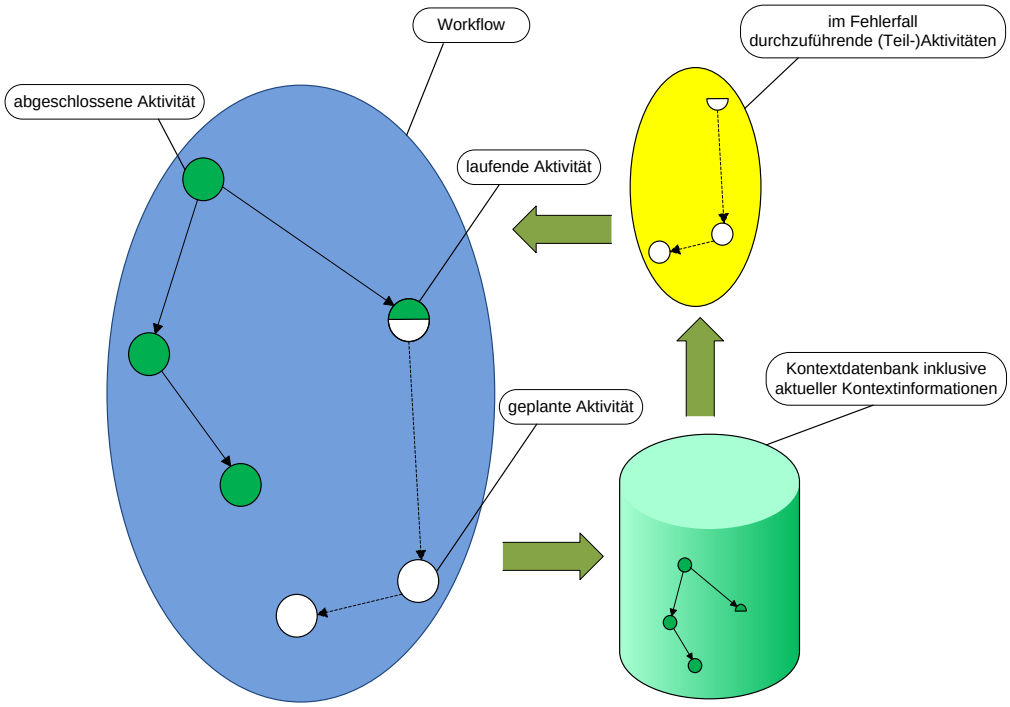

Abbildung 3.2: Kontextbasierte Recovery (nach [LR00])

Im Gegensatz zu mobilen Systemen gibt es in Workflow-Systemen häufig zuverlässige, drahtgebundene Kommunikationskanäle sowie deutlich verlässlichere und leistungsfähigere stationäre Endgeräte, die inklusive ihrer Fähigkeiten a priori bekannt sind. Des Weiteren kann im Normalfall jeder Rechner mit jedem anderen im Netzwerk direkt kommunizieren und Fehler sind eher Ausnahmesituationen. Die persistente Speicherung von Kontextinformationen gehört in mobilen Systemen zu den zentralen Problemen, da je nach Art des entsprechenden mobilen Systems häufig kein zuverlässiger stationärer Server bzw. eine Serverfarm vorhanden ist, die für alle Knoten zu jedem Zeitpunkt erreichbar ist, um periodisch Kontextinformationen auszutauschen bzw. zu speichern. Des Weiteren existiert zum Beispiel in MANETs keine zentrale Steuerungseinheit, die im Falle eines Fehlers die notwendigen Recovery-Schritte auslöst. Eine Adaption dieser Art der Recovery erfordert folglich eine Anpassung an das jeweilige Szenario, also unter anderem an die Art und Architektur des mobilen Systems.

3.4.2 Logging-Mechanismen

In diesem Abschnitt werden gängige Logging-Mechanismen im Bereich der verteilten Systeme betrachtet. Der Schwerpunkt liegt hierbei auf dem Logging von Prozessen.

Allgemeine Betrachtung des Loggings

Unter einem *Logfile* wird eine Datei verstanden, in der Vorgänge innerhalb eines Computers oder Systems protokolliert werden. [Zwa05] Ein *Logfile* ist hierbei zumeist eine ein-

fache Datei, in der Daten sequenziell gespeichert werden, indem neue Daten jeweils ans Ende der Datei angehängt, also keine Einträge überschrieben werden. [FC87] Der Vorgang der Aufzeichnung relevanter Aktivitäten wird hierbei als *Logging* bezeichnet. Welche Aktivitäten relevant sind und auf welche Art und Weise sie aufgezeichnet werden, hängt hierbei vom jeweiligen Anwendungsgebiet und den mit dem Logging verfolgten Zielen ab. In Datenbanksystemen wird die Protokollierung von Vorgängen beispielsweise verwendet, um auf Basis der aufgezeichneten Logdateien die Konsistenz der Datenbank bei Bedarf wieder herzustellen. Andere Anwendungskontexte verwenden Logging hingegen, um die Benutzerfreundlichkeit von Programmen zu gewährleisten, Fehler im System aufzudecken und zu beheben, zur Nebenläufigkeitskontrolle oder zu Sicherheitszwecken (um beispielsweise Angriffe auf ein System aufzuzeichnen und zu verhindern). Logging kann demnach verwendet werden, um fehlertolerante verteilte Systeme zu realisieren. [Ruf94]

Da im Kontext des Managements mobiler Prozesse das Logging von Prozessen im Vordergrund steht, wird im Folgenden das Gebiet des Workflow-Managements aufgrund der bereits genannten Analogien unter dem Aspekt des Loggings von Prozessen näher betrachtet.

Logging in Workflow-Managementsystemen

Im Workflow-Management wird Logging verwendet, um Laufzeit- und Zustandsinformationen und -übergänge von Prozessen festzuhalten. Dieser Vorgang wird auch als *Auditing* bezeichnet. Auditing umfasst die Speicherung aller relevanten Ereignisse im Lebenszyklus eines Prozesses, Subprozesses oder einer Aktivität in einer speziellen Protokolldatei (auch *Audit-Trail* genannt). [LR00] Ereignisse sind in diesem Kontext zum Beispiel der Start, das Ende oder der Abbruch einer Aktivität oder eines Prozesses. Ein Eintrag in einem Audit Trail beinhaltet je nach Einssatzszenario und Logging-Schwerpunkt unterschiedliche Informationen, wobei die folgenden Felder häufig verwendet werden [LR00]:

- Datum und Zeitpunkt eines Ereignisses.
- Identifikator, der ein Ereignis eindeutig charakterisiert (z. B. Start oder Ende einer Aktivität).
- Identifikator, der das jeweilige Ereignis einem Prozess oder Subprozess eindeutig zuordnet.
- Identifikator des Anforderers der jeweiligen Aktion (z. B. das WFMS selbst oder ein bestimmter Benutzer).
- Identifikator der Aktivität, für die das Ereignis geschrieben wurde.

- Weitere Identifikatoren, beispielsweise der Identifikator eines Subprozesses, sofern das Ereignis der Start einer Aktivität ist, die als Subprozess implementiert wurde.

Um Systemunabhängigkeit sowie Interoperabilität zwischen verschiedenen Workflow-Managementsystemen zu gewährleisten und damit auch unternehmensübergreifende Analysen zu ermöglichen, wurde von der *Workflow Management Coalition* (WFMC) der Aufbau der Audit Trails im *Workflow Management Coalition Interface 5* [Wor98] standardisiert. [Mel07] [ZM04] Die im Standard beschriebene Datenstruktur besteht aus einem *Prefix* einem *Body* und einem *Suffix*. Der *Prefix* beinhaltet hierbei den Identifikator der Prozessinstanz, die zum Audit Trail Eintrag geführt hat. Der *Body* variiert in Abhängigkeit von der Art des Ereignisses, wobei der Standard zwischen prozessrelevanten, aktivitätsrelevanten und „work item"-bezogenen Ereignissen unterscheidet. Der *Suffix* ist optional und ermöglicht händlerspezifische Erweiterungen des Datenmodells. [ZM04] Abbildung 3.3 verdeutlicht die vorgeschlagene Datenstruktur der WFMC. Die mithilfe des Auditings aufgezeichneten Informationen können unter anderem für folgende Zwecke verwendet werden: ([KND98] [MR00] [Mel07] [LR00])

- **Bussiness Process Reeingeniering:** Indem Informationen über sämtliche laufende und bereits abgelaufene Prozesse gespeichert und ggf. so aufbereitet werden, dass sie für mannigfaltige Analysen verwendbar sind, können Geschäftsprozesse optimiert und an sich verändernde Gegebenheiten angepasst werden.
- **Recovery:** Laufzeit und Zustandsinformationen können verwendet werden, um auf eventuell auftretende Fehler adäquat zu reagieren und damit Prozesse wiederherzustellen.
- **Aufgabenzuordnung:** In manchen Situationen bzw. bei manchen Prozessen kann es notwendig sein, dass potenziell aufeinander folgende Aktivitäten von unterschiedlichen Personen bzw. der gleichen Person bearbeitet werden. Damit eine derartige Zuordnung möglich ist, sind Informationen über vergangene Prozesse und Aktivitäten essenziell.
- **Rechtliche Gründe:** Aus rechtlichen Gründen kann es notwendig sein, dass auch nach der Ausführung von Prozessen zweifelsfrei feststellbar ist, wer welche Aktivität zu welchem Zeitpunkt durchgeführt hat. In der Luftfahrt ist es zum Beispiel Plicht, dass der komplette Lebenszyklus eines Prozesses für 30 Jahre aufbewahrt wird.
- **Lastausgleich:** Informationen darüber, wer derzeit welche Prozesse bzw. Aktivitäten bearbeitet, können zur Laufzeit verwendet werden, um Lastausgleich zu betreiben, indem Aufgaben z. B. umdelegiert werden.

In Abhängigkeit davon, ob die ermittelten Daten zur Laufzeit oder erst in einer nachträglichen Analyse verwendet werden, spricht man von *Workflow Monitoring* bzw. in letzterem Fall vom *Workflow Controlling*. [MR00] Die persistente Speicherung der Audit Trails

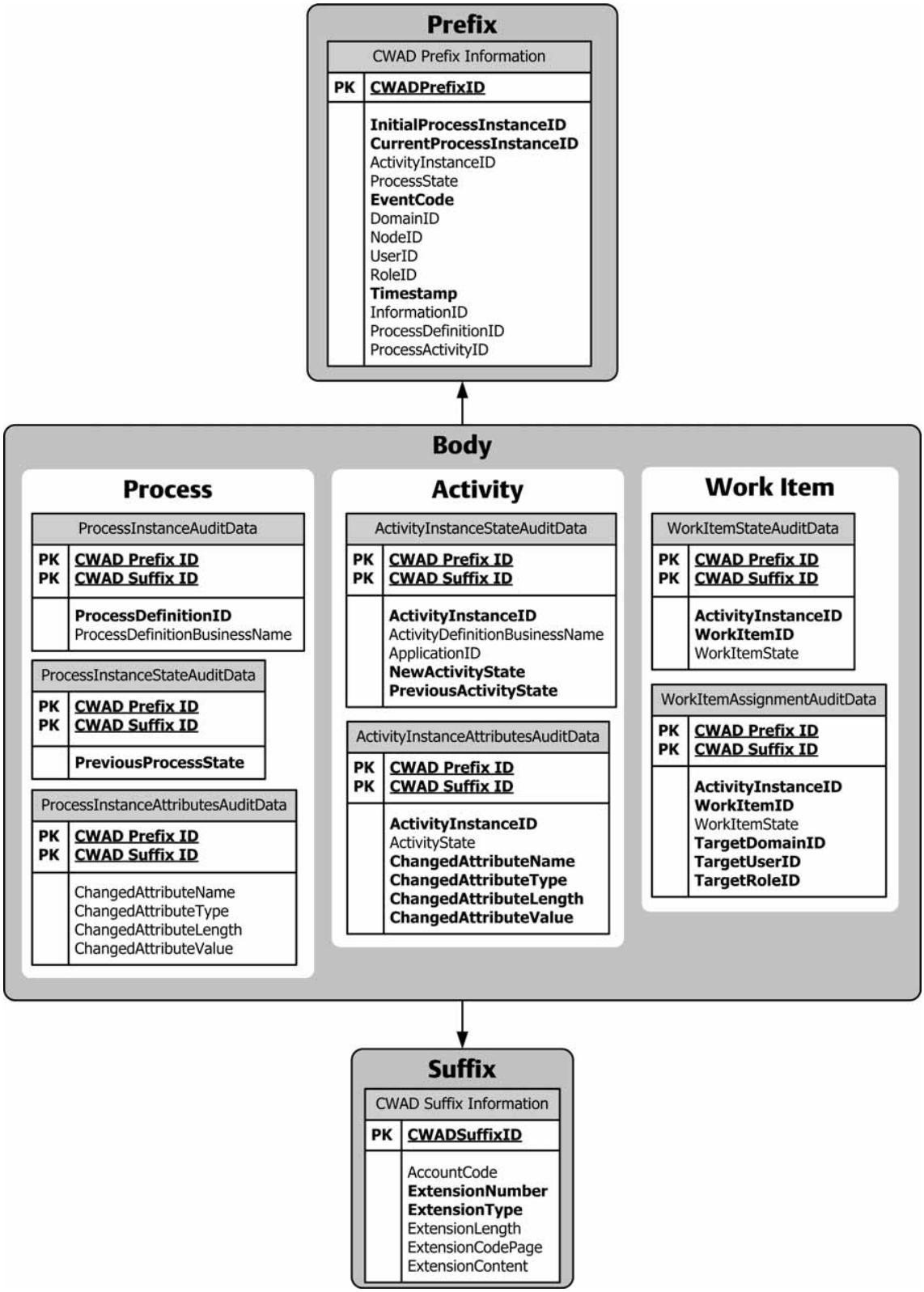

Abbildung 3.3: Audit Trail Data Structure of WfMC Interface 5 (aus [Wor98])

übernimmt in der Regel die Workflow Engine des Workflow-Managementsystems, wobei die Daten selbst häufig in einer Datenbank gespeichert werden. [ZM04]

Betrachtung im Kontext mobiler Prozesse

Eine derartige Überwachung funktioniert im Bereich der Workflows relativ problemlos, da Workflows meistens auf zuverlässigen, schnellen Unternehmensrechnern in einem drahtgebundenen Netzwerk ausgeführt werden und auch a priori sämtliche Knoten im Netzwerk inklusive ihrer Fähigkeiten bekannt sind. Da das WFMS permanent die Kontrolle über den Prozess hat und seine Ausführung leitet, lassen sich die erhobenen Auditing Daten auch recht einfach aufzeichnen, zentral speichern und bei Bedarf analysieren. Im Kontext mobiler Prozesse existiert ggf. keine zentrale Speichermöglichkeit und auch keine zentrale Steuerungseinheit, die permanent die Kontrolle hat. Im Gegenteil wechselt die Kontrolle migrationsbasiert ggf. häufig, des Weiteren sind die Endgeräte unzuverlässiger und es ist a priori auch keineswegs bekannt, welches Endgerät zu welchem Zeitpunkt im Netzwerk vorhanden ist, weshalb die sichere Speicherung der Logdateien in mobilen unzuverlässigen Systemen deutlich problematischer ist.

3.4.3 Monitoring-Mechanismen

Der folgende Abschnitt beschäftigt sich mit dem Aspekt des Monitorings. Neben einer kurzen Einführung in das Thema wird hierbei vor allem der Aspekt der Ausfallerkennung näher betrachtet.

Allgemeine Betrachtung des Monitorings

Für den weitläufigen Begriff *Monitoring* existieren mannigfaltige Definitionen und Anwendungsgebiete. Nach *Nissen* [Nis08] beinhaltet der Begriff Monitoring die Überwachung und Protokollierung des Verhaltens eines Systems, wobei die Auswertung, die Visualisierung der gesammelten Daten sowie die Steuerung, also aktive Beeinflussung des Systems, ebenfalls häufig zu den Aufgaben des Monitorings gezählt wird. Gängige Anwendungsgebiete sind neben Softwaresystemen auch Ökosysteme oder die Überwachung und Steuerung von Geschäftsprozessen in Unternehmen. [Nis08] Eine weitere, speziell auf das Monitoring von Prozessen in verteilten (Software-)Systemen ausgerichtete Definition besagt:

> *„The monitoring of distributed systems involves the collection, interpretation, and display of information concerning the interactions among concurrently executing processes. This information and its display can support the debugging, testing, performance evaluation, and dynamic documentation of distributed systems.“* [JLSU87, S.121]

Die zuletzt genannte Definition geht hierbei im Gegensatz zur erstgenannten allgemeineren Definition neben den Aufgabenbereichen auch konkret auf gängige Ziele des Monitorings (in Softwaresystemen) ein, also unter anderem auf die Performanzanalyse und Fehlererkennung und -behebung ein. Im Kontext dieser Arbeit, also im Management mobiler Prozesse, wird unter dem Begriff Monitoring die Überwachung und Steuerung der Ausführung mobiler Prozesse verstanden. Neben der Fehlererkennung, die hohe Priorität besitzt, gehört unter anderem die Feststellung darüber, wo (also auf welchem Endgerät) sich ein spezifischer mobiler Prozess derzeit befindet und wie weit die Bearbeitung abgeschlossen ist, zu den Aufgabenbereichen (vgl. 3.2.4). Bevor nun konkrete Monitoring-Ansätze betrachtet werden, werden zunächst die unterschiedlichen Arten des Monitorings verdeutlicht.

Arten des Monitorings

Beim Monitoring kann man zwischen *aktivem* und *passivem Monitoring* unterscheiden. Man spricht von *passivem Monitoring*, wenn das jeweilige System Informationen über seinen Status oder Zustand nur per Anfrage preisgibt. Beim *aktiven Monitoring* übermittelt das System ungefragt seine Zustände selbstständig. [MR00] Je nach Art des Systems bietet sich eher die eine oder andere Variante an, so ist es zum Beispiel sinnvoll, aktives Monitoring zu betreiben, wenn das jeweilige System selbst erkennt, welche Informationen relevant sind oder wann Informationen gebraucht werden. Ist dies nicht der Fall, so ist passives Monitoring zumeist angebrachter.

Nach [MS95] lässt sich Monitoring weiterhin danach unterscheiden, ob es *zeit-* oder *eventbasiert* ist. *Zeitbasiertes Monitoring* liegt dann vor, wenn periodisch zu bestimmten Zeitpunkten Informationen über den Status des Systems gesammelt werden. Beim *eventbasierten Monitoring* werden hingegen *Events* also Ereignisse spezifiziert, die für den jeweiligen Betrachter relevant sind und zu denen Statusinformationen aufgezeichnet werden. Da bei der letztgenannten Variante lediglich Statusinformationen zu relevanten Ereignissen aufgezeichnet werden, ist die produzierte Datenmenge im Gegensatz zum zeitbasierten Monitoring, bei dem unabhängig davon, ob der jeweilige Zustand interessant ist, periodisch Aufzeichnungen gemacht werden, häufig deutlich geringer.

Monitoring in Workflow-Managementsystemen

Das *Workflow Monitoring* (auch *Operative Process Controlling*) beschäftigt sich mit der Analyse und Darstellung von Workflow-Instanzen zur Laufzeit. Mithilfe von Monitoring-Informationen wird es Workflow-Administratoren und/oder Prozessmanagern ermöglicht, das Verhalten der Workflow-Instanzen zur Laufzeit zu beeinflussen und zum Beispiel Probleme frühzeitig zu vermeiden oder Fehler zu beheben. Sowohl passives als auch aktives Monitoring kann hierbei je nach Analyseschwerpunkt und Ziel sinnvoll sein.

Aktives Monitoring der Zustände von Workflow-Instanzen wird zum Beispiel im Kontext des Workflow-Managements verwendet, um frühzeitig Warnungen für Aktivitäten oder ganze Workflows zu generieren, die mit hoher Wahrscheinlichkeit ihre Deadline nicht einhalten können oder um automatisch Fehlerberichte für Workflows zu schreiben, die ihre Deadline bereits überschritten haben. Passives Monitoring kann hingegen verwendet werden, um Kunden über den Status ihrer Bestellung zu informieren oder festzustellen, wer derzeit eine bestimmte Aktivität innerhalb eines Workflows bearbeitet.

Im Bereich des Workflow-Managements kann darüber hinaus zwischen *technischem* und *organisatorischem* Monitoring unterschieden werden. Der Erstgenannte beschäftigt sich mit der Performanzanalyse (z. B. Systemlast verbessern), während der Letztgenannte dazu dient, die Effizienz innerhalb der Organisation zu verbessern (z. B. Wartezeiten verringern). Die Überwachung nach den genannten Kriterien wird hierbei im Allgemeinen mithilfe von Logdateien, den so genannten Audit Trails (vgl. 3.4.2) durchgeführt, was bezüglich der Adaption dieser Mechanismen für mobile Prozesse wieder zu den am Ende des Abschnitts „Logging in Workflow-Managementsystemen“ beschriebenen Problemen führt. [MR00] [ZM04]

Ausfallerkennung in verteilten Systemen

Der Ausfall eines Knotens führt im Kontext der mobilen Prozesse in der Regel dazu, dass die Ausführung sämtlicher sich zum Zeitpunkt des Ausfalls auf dem Knoten befindender Prozesse beeinträchtigt wird. Um derartige Fehler zu erkennen und die zugehörigen Prozesse wieder herzustellen, ist eine möglichst zuverlässige Ausfallerkennung essenziell. Im Folgenden werden daher Algorithmen betrachtet, die die Ausfallerkennung in verteilten Systemen zum Ziel haben.

Unterschiedliche Ausfallerkennungsalgorithmen werden häufig durch ihren Grad an *accuracy* (Genauigkeit) und ihren Grad an *completeness* (Vollständigkeit) charakterisiert. *Accuracy* bedeutet hierbei, dass keine lebendigen Prozesse bzw. Knoten verdächtigt werden und *completeness*, dass sämtliche ausgefallenen Prozesse bzw. Knoten erkannt werden. Jede der beiden Eigenschaften lässt sich trivialerweise isoliert umsetzen, indem keiner bzw. alle verdächtigt werden. [FT05] Die deterministische Umsetzung beider Eigenschaften ist in asynchronen Systemen allerdings selbst dann, wenn nur schwache Umsetzungsformen der Eigenschaften verlangt werden, nicht möglich. [CT96]

Heartbeat failure detectors: Ein häufig in der Praxis verwendeter Algorithmus zur Erkennung von Knotenausfällen in verteilten Systemen basiert auf so genannten *Heartbeats*. *Heartbeats* sind Nachrichten, die im traditionellen Ansatz jeder Knoten jedem anderen Knoten periodisch zum Zwecke der Ausfallerkennung übermittelt. Sofern ein Knoten p nach einem festgelegten Zeitintervall keine neue Heartbeat-Nachricht von einem Knoten q bekommt, verdächtigt er den Knoten des Ausfalls. [CTA00] Der Begriff „verdächtigt“ verdeutlicht hierbei bereits, dass der überwachende Knoten hierbei keineswegs un-

terscheiden kann, ob der überwachte Knoten tatsächlich ausgefallen oder einfach nur sehr langsam ist. [BMS02] Gerade in MANETs kann es aufgrund der fehlerbehafteten Kommunikation und hohen Mobilität sowie der hieraus resultierenden Übertragungsverzögerung daher dazu kommen, dass eigentlich funktionstüchtige Knoten verdächtigt werden. Da ein Knoten einen anderen fälschlicherweise verdächtigen kann, spricht man in diesem Kontext auch von so genannten *unreliable failure detectors*. [EB07] *Tourag et al.* haben gezeigt, dass das *Konsensproblem* auch mit unreliable failure detectors, die eine endliche Menge an Fehlern machen, lösbar ist. [CT96]

Der hier aufgeführte klassische Heartbeat-Algorithmus hat zwei grundlegende Nachteile. Einer der beiden Nachteile ist, dass die Fehler bzw. die Ausfallerkennung von der letzten Heartbeat-Nachricht abhängig ist und daher ggf. die Genauigkeit der Fehlererkennung negativ beeinflusst. Der zweite Nachteil ist, dass das Timeout-Intervall fest ist und damit die Netzwerklast nicht berücksichtigt wird, was dazu führen kann, dass funktionierende Knoten fälschlicherweise aufgrund hoher Netzwerklast und der damit verbundenen langsameren Nachrichtenübermittlung als fehlerhaft erkannt werden.

Chen et al. [CTA00] schlagen zur Verbesserung des klassischen oben aufgeführten Heartbeat-Algorithmus die Verwendung von so genannten *Freshness-Points* vor. Ein *Fresshness-Point* π_i^v ist hierbei eine Schätzung der Ankunftszeit der i-ten Heartbeat-Nachricht des Knotens v anhand bereits empfangener Heartbeat-Nachrichten, wodurch das erstgenannte Problem behoben und damit die Genauigkeit der Fehlererkennung verbessert wird.

Bertier et al. [BMS02] entwickelten wiederum auf Basis der Erkenntnisse von *Chen et al.* [CTA00] einen adaptiven Heartbeat-Algorithmus, in dem die Sendeperiode der Heartbeats (Zeit zwischen zwei Heartbeat-Nachrichten) als Funktion der Dienstgüte des Netzwerkes und den jeweiligen Anwendungsanforderungen ständig adaptiert wird. [EB07] Bei ihrem Algorithmus wird die Ankunftszeit der nächsten Heartbeat-Nachricht ähnlich wie bei *Chen et al.* basierend auf älteren Heartbeat-Nachrichten geschätzt, allerdings wird im Gegensatz zu *Chen et al.* keine konstante *safety margin* (Sicherheitsspielraum), sondern eine dynamisch berechnete hinzu addiert. Die *safety margin* ist hierbei dazu da, um zu verhindern, dass aktive Knoten aufgrund von Übertragungsverzögerungen oder Prozessorüberlastungen fälschlicherweise des Ausfalls verdächtigt werden. [BMS02] Sie wird dynamisch mithilfe des Algorithmus von *Jacobson* [PA00] unter der Annahme, dass das Systemverhalten nicht konstant ist, bestimmt. Das zu Grunde liegende Systemmodell, nach dem *Bertier et al.* [BMS02] ihren Algorithmus entwickelt haben, besteht aus einem verteilten System, in dem eine feste Anzahl an Prozessen drahtgebunden und nachrichtenbasiert über ein LAN miteinander kommuniziert, wobei Prozesse nur durch einen Crash ausfallen können und der Ausfall permanent ist. Dieses Modell entspricht per Definition nicht den Gegebenheiten in mobilen Ad-hoc-Netzen, weshalb für mobile Systeme Anpassungen bzw. angepasste Algorithmen notwendig sind.

Aus diesem Grund haben *Elhadef et al.* [EB07] einen *Gossip* Algorithmus für mobile Adhoc-Netze unter anderem basierend auf den vorangehenden Algorithmen von *Bertier et*

al. [BMS02], *Jacobson* [PA00] und *Chen et al.* [CTA00] entwickelt. *Gossip* bedeutet, dass Heartbeat-Nachrichten lediglich an einige Nachbarn und nicht an alle Knoten gesendet werden, was den Nachrichtenoverhead und damit auch den Energieverbrauch der mobilen Endgeräte deutlich reduziert. [FT05] Jeder Knoten sendet periodisch Heartbeat-Nachrichten an sämtliche, direkt erreichbare Knoten (Nachbarn), wobei die Informationen anderer Heartbeats ebenfalls auf Basis eines Vektors oder Arrays mitübermittelt werden. Jede Stelle im Vektor beinhaltet hierbei den höchsten bekannten Heartbeat vom korrespondierenden Knoten. Wird von einem Knoten folglich ein Heartbeat empfangen, so werden die korrespondierenden Einträge im Vektor auf das Maximum des lokalen und des empfangenen Vektors erhöht. Die transitive Natur dieses Algorithmus führt daher im Optimum dazu, dass jeder Knoten von jedem anderen indirekt oder direkt die benötigten Informationen bekommt. Wird nach einer bestimmten Zeit kein (direkter oder indirekter) Heartbeat empfangen, so verdächtigt der jeweilige Knoten den anderen. Das Systemmodell von *Elhadef et al.* [EB07] sieht bei seinem Algorithmus eine feste Anzahl an Knoten vor, die zufällig in einem Bereich verteilt sind, die Möglichkeit haben, sich (langsam) zu bewegen und nach den Prinzipien in MANETs drahtlos zu interagieren, wobei die Übertragungsreichweite beschränkt ist. Die einzelnen Knoten können ausfallen, wobei Ausfälle auch in diesem Systemmodell permanent sind. [EB07] Insgesamt entspricht das Modell am ehesten dem Modell, das auch dieser Arbeit zu Grunde liegt, allerdings mit der Ausnahme, dass die Anzahl der Knoten nicht fest ist und auch Ausfälle nicht zwingend permanent sind.

Pinging failure detectors: Weitere Möglichkeiten zur Ausfallerkennung bieten die so genannten *pinging failure detectors*. Bei diesem Konzept schickt ein Prozess p einem anderen Prozess q periodisch Anfrage-Nachrichten, auf die q mit einer „Lebenszeichen"-Nachricht antwortet. Sofern der Prozess q nicht innerhalb eines bestimmten Zeitintervalls antwortet, verdächtigt p ihn des Ausfalls. Dieser Ansatz hat insgesamt einige gravierende Nachteile gegenüber dem Heartbeat-Ansatz: Zum einen müssen hier doppelt so viele Nachrichten versendet werden, zum anderen muss hier nicht nur die Ankunft einer Nachricht abgeschätzt werden, sondern die Übertragungsverzögerung der Anfrage, die Reaktionszeit und die Übertragungsverzögerung der Antwort. Insgesamt ist dieser Ansatz folglich deutlich unterlegen und wird daher nicht näher betrachtet. [BMS02]

Betrachtung im Kontext mobiler Prozesse:

In diesem Teilabschnitt wurden diverse Methoden und Algorithmen zur Ausfallerkennung vorgestellt. Es hat sich hierbei gezeigt, dass die *pinging failure detectors* den adaptierten *Heartbeat failure detectors* in der Regel unterlegen sind. Für die Ausfallerkennung im Kontext mobiler Prozesse bietet sich also ein angepasster adaptiver Heartbeat-Algorithmus an. Wie die Anpassungen im Detail aussehen, hängt unter anderem von der

Art und Architektur des mobilen Systems ab, die im folgenden Abschnitt näher betrachtet wird.

3.4.4 Fazit

In diesem Abschnitt wurden bestehende Recovery-, Logging- und Monitoring-Ansätze unter dem Aspekt der Adaption für mobile Prozesse betrachtet. In allen drei Gebieten wurden durchaus brauchbare Ansätze gefunden, wobei gerade aufgrund der per Definition vorhandenen Mobilität in mobilen Systemen und den hieraus resultierenden Beschränkungen Anpassungen notwendig sind. Wie diese Anpassungen aussehen müssen, hängt dabei unter anderem davon ab, wie die Architektur des jeweiligen mobilen Systems beschaffen ist, d.h., ob beispielsweise eine Infrastruktur vorhanden ist, die für die persistente Speicherung von Logdaten genutzt werden kann oder ob das jeweilige System vollkommen infrastrukturlos und selbstorganisierend (wie z. B. in MANETs) ist. Im nächsten Abschnitt wird der Aspekt der Architektur unter Berücksichtigung der Anforderungen an eine Managementinfrastruktur näher betrachtet und es wird evaluiert, welche Ausprägungsform (infrastrukturlos oder infrastrukturbasiert) für das Management mobiler Prozesse besser geeignet ist.

3.5 Vergleich und Bewertung möglicher Architekturen

Nachdem definiert wurde, welche Funktionalität das Management mobiler Prozesse umfassen sollte und welche Anforderungen für eine solche Managementinfrastruktur gelten, werden im Folgenden die grundlegenden Umsetzungsmöglichkeiten verglichen. Hierbei werden sowohl infrastrukturbasierte als auch infrastrukturlose Ansätze evaluiert, um anhand des Anforderungskataloges zu entscheiden, auf welche Art und Weise das Managementsystem umgesetzt werden sollte.

3.5.1 Infrastrukturbasierte Umsetzung

In diesem Abschnitt werden zunächst zwei infrastrukturbasierte Ansätze diskutiert sowie deren Vor- und Nachteile herausgestellt. Beide Varianten entsprechen dem *Client-Server-Modell*, d.h., die einzelnen auf den Endgeräten laufenden Programme werden im verteilten System in die zwei Gruppen *Server* (Diensterbringer) und *Client* (Dienstnehmer) eingeteilt. Ein *Server* ist hierbei als ein Softwareprogramm zu verstehen, das einen Dienst implementiert und bereitstellt (in diesem Kontext die Managementfunktionalität), während ein *Client* wiederum ein Softwareprogramm ist, das den jeweiligen Dienst anfordert bzw. nutzt, indem es eine Anfrage sendet und auf die Antwort des Servers wartet. [TS08] Beide hier vorgestellten Varianten unterscheiden sich lediglich im Hinblick auf die Anzahl der Server.

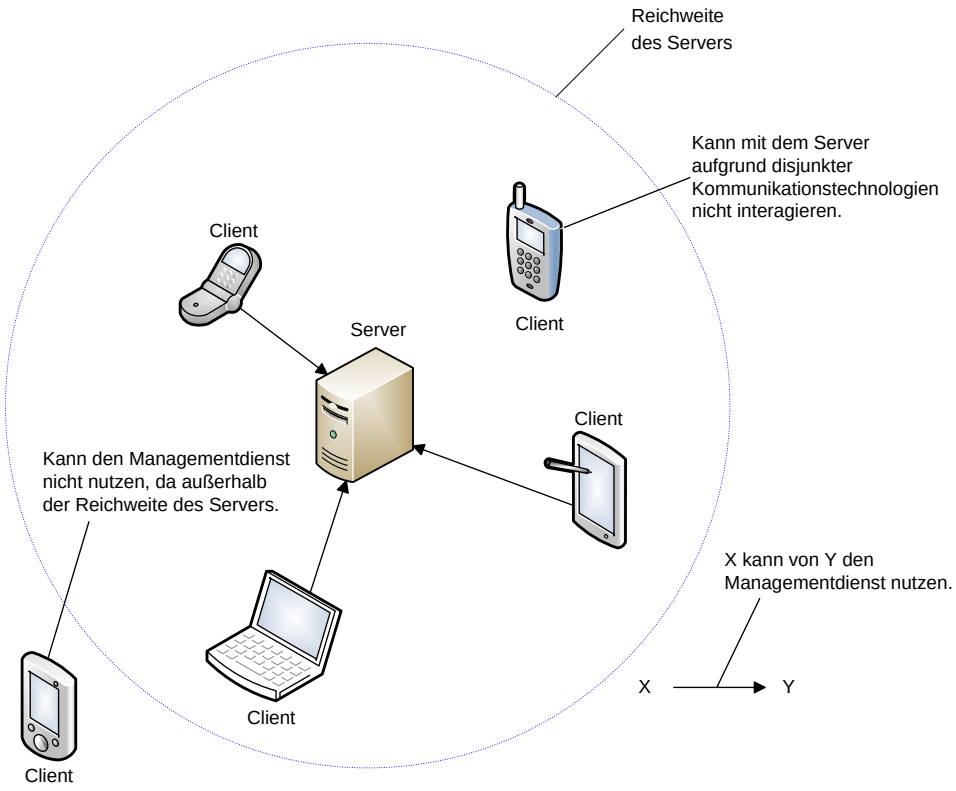

Abbildung 3.4: Ein-Server-Variante

Ein-Server-Variante

Wie der Name bereits verdeutlicht, ist diese Variante so aufgebaut, dass ein ausgezeichneter, stationärer Server verwendet wird, der die komplette Managementfunktionalität als Dienst bereitstellt. Sämtliche andere Endgeräte im Netzwerk agieren folglich potenziell als Clients, die die zugehörige Managementfunktionalität in Anspruch nehmen können, Abbildung 3.4 verdeutlicht die Variante grafisch.

Vorteile

- **Koordinationsaufwand:** Der Koordinationsaufwand zwischen mehreren Servern zur Bereitstellung der Managementfunktionalität entfällt, da per Definition ein einziger Server sämtliche Aufgaben übernimmt.
- **Auswahlalgorithmus:** Da ein spezieller Server von vornherein für die Managementaufgaben zuständig ist, wird kein Auswahlalgorithmus benötigt, um zu bestimmen, welches Gerät das Management übernimmt.
- **Energie:** Da der Server stationär ist, lässt er sich in den meisten Szenarien mit Strom aus der Steckdose versorgen, weshalb Ausfälle unwahrscheinlicher als bei mobilen Endgeräten sind. Dies hat folglich positive Effekte auf Aspekte wie Verfügbarkeit und Zuverlässigkeit.

Nachteile

- **Fehlertoleranz:** Bei diesem Ansatz führt der Ausfall des Management-Servers zum Ausfall des Managementsystems und damit der kompletten zugehörigen Funktionalität.
- **Skalierbarkeit:** Ab einer gewissen Anzahl an Endgeräten, die die Managementfunktionalität nutzen möchten, bekommt ein einzelner Server Probleme mit der Beantwortung der Anfragen und der Bearbeitung seiner Aufgaben. Dies führt zu negativen Effekten in Bezug auf Verfügbarkeit und Zuverlässigkeit.
- **Kosten und Verantwortlichkeit:** Damit diese Variante funktioniert, ist ein zentraler Server notwendig, d.h., einerseits sind zusätzliche Kosten bezüglich der Infrastruktur inhärent, andererseits muss es eine „Partei“ geben, die sich dafür verantwortlich fühlt, einen Server bereitzustellen.
- **Einsatzszenario:** Dieser Ansatz ist in Gebieten, in denen kein Server aufgestellt werden kann, nicht anwendbar.
- **Kommunikationsfähigkeit:** Aufgrund der hohen Heterogenität bezüglich der Kommunikationsprotokolle und -technologien insbesondere bei der drahtlosen Kommunikation (vgl. 2.2.6) müsste ein Server sämtliche Technologien, also unter anderem IrDa, Bluetooth und WLAN beherrschen, damit jedes Endgerät die Möglichkeit hat, die Managementfunktionalität zu nutzen.
- **Reichweite:** Da jede der verschiedenen Kommunikationstechnologien lediglich eine beschränkte Reichweite hat, kann mit einem einzigen Management-Server auch nur ein beschränktes Gebiet mit Managementfunktionalität versorgt werden.

Multiple-Server-Variante

Die Managementfunktionalität wird in dieser Umsetzungsart durch mehrere stationäre, kooperierende, ausgezeichnete Server als Dienst bereitgestellt (Abbildung 3.5).

Vorteile

- **Fehlertoleranz:** Wird ein Managementsystem durch eine größere Anzahl an kooperierenden Servern bereitgestellt, dann besteht die Möglichkeit, dass einzelne Ausfälle kompensiert werden können, was zu einer höheren Ausfallsicherheit des Systems und damit auch zu einer größeren Fehlertoleranz führt.
- **Skalierbarkeit:** Mehrere Server können im Gegensatz zu weniger Servern der gleichen Bauart mehr Aufgaben und damit auch mehrere Endgeräte mit der zugehörigen Managementfunktionalität versorgen, wobei festzuhalten ist, dass eine hohe Anzahl an Servern natürlich auch eine Belastungsgrenze hat. Das Hinzufügen

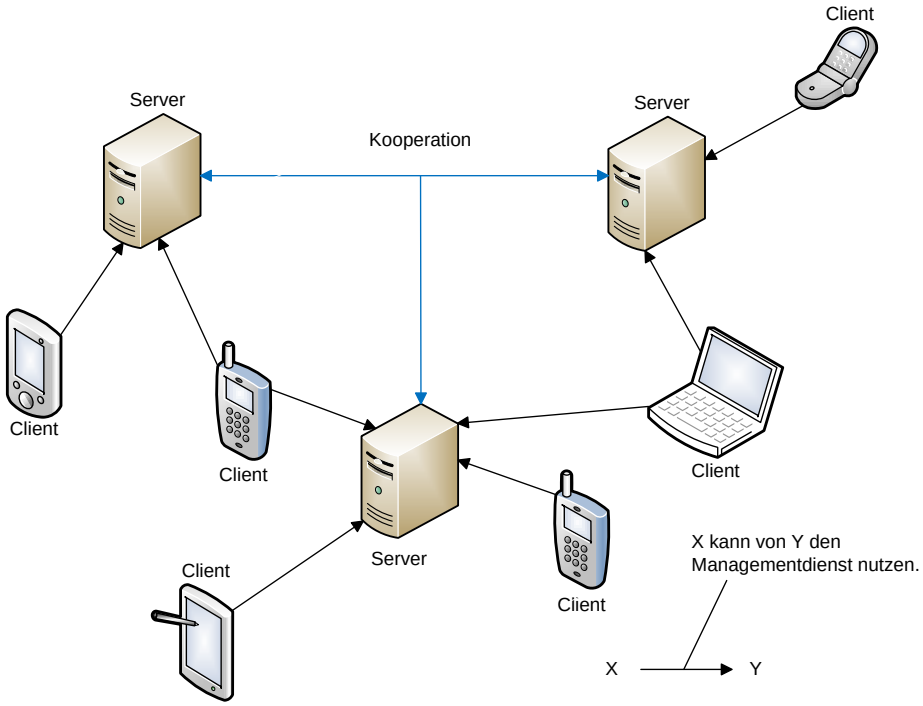

Abbildung 3.5: Multiple-Server-Variante

weiterer Server in Abhängigkeit der Benutzeranzahl ist des Weiteren problematischer und aufwändiger als bei infrastrukturlosen Varianten, bei denen im Prinzip einfach ein weiterer Nutzer Managementaufgaben übernehmen kann.

- **Kommunikationsfähigkeit:** In diesem Szenario sind mehrere Server verfügbar, d.h., es ist nicht notwendig, dass ein spezieller Server sämtliche Kommunikationsarten und -technologien unterstützt, sondern dass diese Funktionalität über die einzelnen Server verteilt werden kann. Ein „Alleskönner“ ist hier also nicht notwendig.
- **Energie:** Genau wie bei der Ein-Server-Variante kann auch hier in vielen Szenarien der Strom aus der Steckdose bezogen werden, was eine höhere Ausfallsicherheit als bei mobilen Endgeräten bewirkt.
- **Auswahlalgorithmus:** In dieser Variante steht direkt fest, welche Server die Managementfunktionalität bereitstellen. Ein Auswahlalgorithmus ist demnach nicht notwendig.
- **Reichweite:** Zwar hat jede Kommunikationstechnologie eine beschränkte Reichweite, allerdings können mehrere geografisch verteilte Server im Prinzip ein deutliches größeres Areal mit Managementfunktionalität versorgen.

Nachteile

- **Koordinationsaufwand:** Im Gegensatz zu Varianten mit einem Server ist die Koordination zwischen den Managementknoten notwendig.
- **Einsatzszenario:** Dieser Ansatz ist in Gebieten, in denen kein Server aufgestellt werden kann, nicht anwendbar.
- **Kosten und Verantwortlichkeit:** Damit diese Variante funktioniert, sind mehrere Server notwendig, d.h., einerseits sind zusätzliche Kosten bezüglich der Infrastruktur inhärent, andererseits muss es eine „Partei“ geben, die sich dafür verantwortlich fühlt, Server bereitzustellen.

Zwischenfazit infrastrukturbasierte Varianten

Im direkten Vergleich der beiden Varianten ist letztere, also die Multiple-Server-Variante, aufgrund der höheren Punkteanzahl auf der Pro-Seite und der geringeren Punkteanzahl auf der Kontra-Seite der ersteren Variante vorzuziehen. Fehlertoleranz und Skalierbarkeit sind besonders wichtige Aspekte für ein Managementsystem, weshalb gerade diese beiden Punkte eine höhere Gewichtung verdienen und daher ebenfalls für einen Ansatz mit mehreren Servern sprechen.

3.5.2 Infrastrukturlose Umsetzung

In diesem Abschnitt werden zwei infrastrukturlose Varianten vorgestellt und bezüglich des Anforderungskataloges evaluiert. Infrastrukturlos bedeutet, dass einer oder mehrere Teilnehmer des Netzwerks die Managementfunktionalität zusätzlich bereitstellen und damit keine zusätzliche Hardware/Server hierfür vorgesehen sind. Dies kann bzw. können im Prinzip auch ein oder mehrere stationäre Server sein, in diesem Fall wird aber zur strikteren Trennung der Umsetzungsmöglichkeiten von einem/mehreren mobilen Endgerät/Endgeräten ausgegangen. Beide in diesem Abschnitt vorgestellten Varianten entsprechen den *Peer-to-Peer*-Prinzipien (P2P), bei denen jeder Teilnehmer sowohl Dienste anbieten als auch nutzen kann, also als Server und als Client (auch *Servent* genannt) gleichzeitig agieren kann. [TS08]

Ein-Servent-Variante

In dieser Variante übernimmt im Gegensatz zu den infrastrukturbasierten Umsetzungsmöglichkeiten kein ausgezeichneter Server, sondern einer der Teilnehmer im Netzwerk zusätzlich die komplette Managementfunktionalität. Diese Variante schränkt das Peer-to-Peer-Modell insofern ein, als dass zu jedem Zeitpunkt lediglich ein Endgerät für alle anderen die Funktionalität bereitstellt und damit als Server und Client gleichzeitig agiert (Abbildung 3.6).

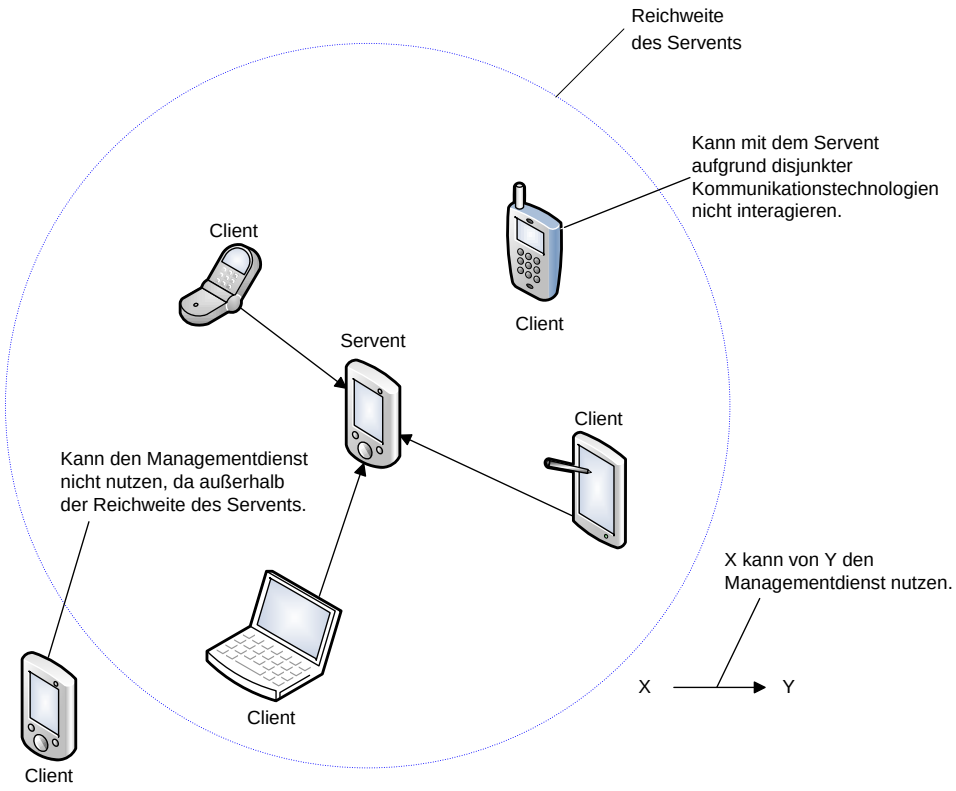

Abbildung 3.6: Ein-Servent-Variante

Vorteile

- **Einsatzszenario:** Dieser Ansatz ist aufgrund seiner Ad-hoc-Charakteristik auch in Gebieten einsetzbar, in denen keine Server aufgestellt werden können.

- **Kosten und Verantwortlichkeit:** Da kein separater Server aufgestellt werden muss, fallen auch keine Kosten an. Des Weiteren wird bezüglich der Verantwortlichkeit auch keine „Partei“ benötigt, die ihn bereitstellt. Trotzdem muss sich wenigstens ein Benutzer bereiterklären, die Managementfunktionalität zusätzlich anzubieten.

- **Koordinationsaufwand:** Da nur ein Teilnehmer die Managementfunktionalität bereitstellt, ist keine Koordination mit anderen Endgeräten notwendig.

Nachteile

- **Fehlertoleranz:** Bei diesem Ansatz führt der Ausfall des Managementgeräts zum Ausfall des Managementsystems und damit der kompletten zugehörigen Funktionalität. Dies kann allerdings durch eine Neuwahl eines neuen Managementgeräts zumindest zum Teil kompensiert werden, sofern das Szenario nicht zu dynamisch und fehleranfällig ist.

- **Skalierbarkeit:** Ab einer gewissen Anzahl an Endgeräten, die die Managementfunktionalität nutzen möchten, bekommt ein einzelnes Gerät Probleme mit der Beantwortung der Anfragen und der Bearbeitung seiner Aufgaben.
- **Kommunikationsfähigkeit:** Damit jedes Endgerät unabhängig von seiner Kommunikationstechnologie prinzipiell die Managementfunktionalität nutzen kann, muss der Management-Server ein „Allrounder“ sein.
- **Energie:** Ein mobiles Endgerät ist in der Regel auf eine Batterie oder einen Akku angewiesen, was zu einer höheren Ausfallwahrscheinlichkeit aufgrund von potenzieller Energiearmut führt. Des Weiteren muss das Endgerät bei steigender Anzahl an Nutzern seiner Funktionalität mehr Aufgaben bewältigen und auch mehr Nachrichten verschicken, was zu einer schnelleren Entladung der Energiequelle führt.
- **Auswahlalgorithmus:** Um zu bestimmen, wer die Managementfunktionalität übernimmt, ist ein Auswahlalgorithmus notwendig, außerdem muss das Ergebnis im Anschluss an alle Teilnehmer kommuniziert werden.
- **Reichweite:** Da jede der verschiedenen Kommunikationstechnologien lediglich eine beschränkte Reichweite hat, kann mit einem einzigen Managementendgerät auch nur ein beschränktes Gebiet mit Managementfunktionalität versorgt werden.

Multiple-Servents-Variante

Potenziell mehrere Teilnehmer innerhalb des Netzwerkes kooperieren, um die Managementfunktionalität bereitzustellen. Abbildung 3.7 verdeutlicht exemplarisch denkbare Beziehungen bzw. Interaktionen zwischen potenziell mobilen Endgeräten eines Netzwerks.

Vorteile

- **Fehlertoleranz:** Wird ein Managementsystem durch eine größere Anzahl an kooperierenden Endgeräten bereitgestellt, so besteht die Möglichkeit, dass Ausfälle einzelner Knoten kompensiert werden können, was zu einer höheren Ausfallsicherheit des Systems und damit auch zu einer größeren Fehlertoleranz führt. Da dieses Modell ohne ausgezeichnete Server auskommt, besteht darüber hinaus die Möglichkeit, den Ausfall eines Managers zu kompensieren, indem aus der Menge der anderen Teilnehmer ein neuer gewählt wird, der seinen Platz einnimmt.
- **Skalierbarkeit:** Da in diesem Szenario mehrere Endgeräte die Managementaufgaben übernehmen, können in der Regel mehrere Endgeräte die zugehörige Funktionalität nutzen. Ein weiterer Vorteil ist, dass es prinzipiell möglich ist, bei Bedarf

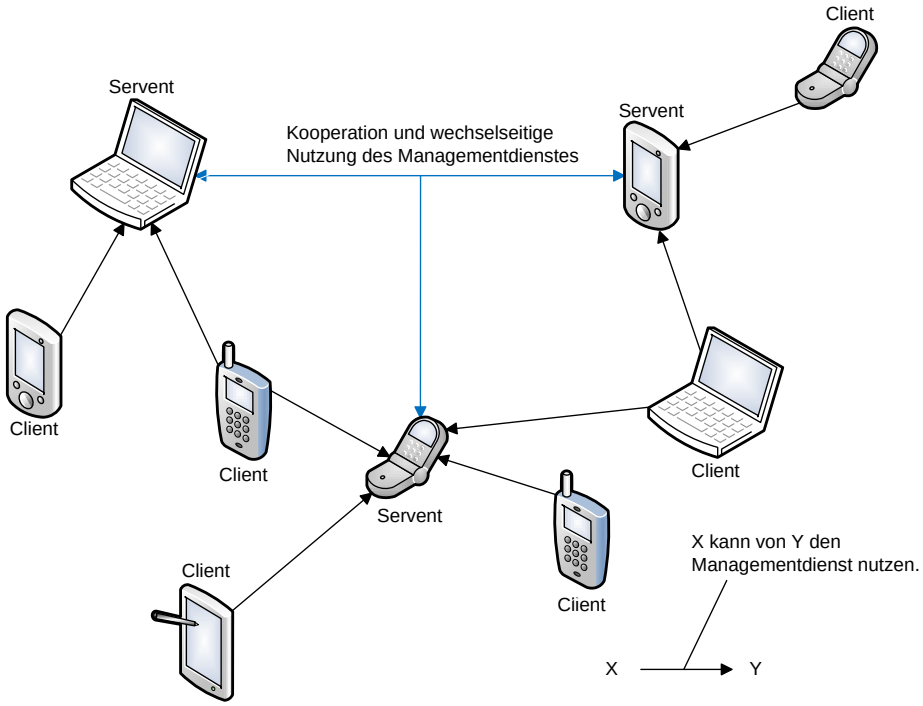

Abbildung 3.7: Multiple-Servents-Variante

weitere Managementknoten aus der Menge der Teilnehmer zu rekrutieren oder Managementknoten abzubauen.

- **Einsatzszenario:** Dieser Ansatz ist aufgrund seiner Ad-hoc-Charakteristik auch in Gebieten einsetzbar, in denen keine Server aufgestellt werden können.

- **Kommunikationsfähigkeit:** Da mehrere Endgeräte an der Erbringung der Managementfunktionalität mitarbeiten, ist bezüglich der Kommunikationstechnologie kein „Allrounder“ notwendig.

- **Kosten und Verantwortlichkeit:** Da kein separater Server aufgestellt werden muss, fallen hier auch keine Kosten an. Des Weiteren wird bezüglich der Verantwortlichkeit auch keine „Partei“ benötigt, die ihn bereitstellt. Damit das Szenario effizient funktioniert, müssen möglichst viele Endgeräte in der Lage sein, als „Manager“ zu agieren und zudem auch bereit sein, diese Rolle auszufüllen.

- **Reichweite:** Zwar hat jede Kommunikationstechnologie lediglich eine beschränkte Reichweite, allerdings können mehrere Endgeräte prinzipiell ein deutliches größeres Areal mit Managementfunktionalität versorgen.

Nachteile

- **Auswahlalgorithmus:** Um zu bestimmen, wer die Managementfunktionalität übernimmt, ist ein Auswahlalgorithmus notwendig, außerdem muss das Ergebnis im Anschluss an alle Teilnehmer kommuniziert werden. Besonders in Szenarien mit hoher Ausfallrate und Mobilität kann es sehr schnell dazu kommen, dass permanent neue Servents akquiriert werden müssen und damit die eigentliche Kooperation beeinträchtigt wird.
- **Energie:** Besteht die Managementgruppe aus sehr vielen mobilen Endgeräten, so gilt auch hier aufgrund der Energierestriktion, dass die Ausfallwahrscheinlichkeit im Gegensatz zu stationären Managern höher ist. Da die einzelnen Aufgaben aber im Allgemeinen über mehrere Endgeräte verteilt werden, ist die Überlastungsgefahr und damit auch die Energieintensität geringer als bei einem einzigen mobilen Servent.
- **Koordinationsaufwand:** Ist die Managementfunktionalität verteilt, so ist Koordination zwischen den einzelnen Erbringern notwendig, was zu einer höheren Kommunikationshäufigkeit und damit auch zu einem höheren Energieverbrauch führt.

Zwischenfazit infrastrukturlose Varianten

Das Verhältnis der aufgeführten Pro- und Kontra-Punkte spricht auch in dieser Variante für einen Ansatz, bei dem die Managementfunktionalität auf mehrere Endgeräte verteilt wird. Die erste aufgeführte Möglichkeit ist im Vergleich mit der anderen infrastrukturlosen Variante gerade aufgrund der schlechten Fehlertoleranz und Skalierbarkeit keine erstrebenswerte Lösung.

3.5.3 Bewertung

Tabelle 3.1 gibt einen Überblick über die gewonnenen Erkenntnisse im Hinblick auf die Vor- und Nachteile der einzelnen Ansätze. Im jeweiligen Zwischenfazit hat sich bereits herausgestellt, dass die Varianten, bei denen die Managementfunktionalität von einem Endgerät bzw. Server erbracht wird, aufgrund der Anforderungen wie Fehlertoleranz und Skalierbarkeit für ein Managementsystem in mobilen, volatilen Umgebungen nicht geeignet sind.

Die infrastrukturbasierte Variante mit mehreren Servern hat gegenüber seinem infrastrukturlosen Pendant die Vorteile, dass aufgrund der zuverlässigeren Energiequelle die Ausfallwahrscheinlichkeit geringer ist und auch kein Auswahlalgorithmus verwendet werden muss, um die Manager zu bestimmen. Zu berücksichtigen bleibt hierbei allerdings, dass aufgrund der beschränkten Energiequelle bei mobilen Endgeräten zwar prinzipiell eine größere Ausfallwahrscheinlichkeit der Einzelgeräte besteht, dafür aber im Falle eines Ausfalls die Möglichkeit besteht, ein neues Endgerät als Manager aus der Menge der anderen im Netzwerk befindlichen Teilnehmer zu rekrutieren und damit den Ausfall zu kompensieren, während dies in infrastrukturbasierten Systemen aufgrund der

	infrastrukturbasiert		*infrastrukturlos*	
Anforderung	**Ein Server**	**Multiple Server**	**Ein Servent**	**Multiple Servents**
Fehlertoleranz	- -	+	- -	++
Skalierbarkeit	- -	+	- -	++
Kosten und Verantwortlichkeit	-	- -	++	++
Einsatzszenario	-	- -	++	++
Kommunikationsfähigkeit	- -	++	- -	++
Auswahlalgorithmus	++	++	-	- -
Energie	++	++	- -	- -
Koordinationsaufwand	++	- -	++	- -
Reichweite	- -	++	- -	+ +

Tabelle 3.1: Vor- und Nachteile der Umsetzungsmöglichkeiten einer Managementkomponente für mobile Prozesse

unterschiedlichen Rollen (Client/Server) nicht möglich ist. Des Weiteren bleibt festzuhalten, dass der Vorteil der zuverlässigeren Energiequelle nur dann gilt, wenn für das Managementsystem nach infrastrukturloser Art mobile Manager gewählt werden.

Der Aspekt der Skalierbarkeit ist in infrastrukturlosen Ansätzen besser, da sich hier leichter Manager hinzufügen bzw. abbauen lassen. Weiterhin hat der Verzicht auf eine Infrastruktur gravierende Vorteile: So besteht unter anderem das Problem der Verantwortlichkeit nicht, es sind keine extra Kosten für die Infrastruktur notwendig und das mögliche Einsatzspektrum ist deutlich größer. Da prinzipiell in der infrastrukturlosen Variante auch stationäre Endgeräte als Manager gewählt werden können, kann man darüber hinaus sagen, dass die infrastrukturbasierte Umsetzungsmöglichkeit ein Spezialfall der infrastrukturlosen Variante ist, bei der alle Manager, die gewählt werden, stationäre, ausgezeichnete Endgeräte sind. Demnach lässt sich eine nach infrastrukturlosen Maßstäben entwickelte Managementkomponente, durchaus auch in ursprünglich für eine infrastrukturbasierte Nutzung konzipierten Systemen verwenden. Der Umkehrschluss gilt nicht, da bei der infrastrukturlosen Variante keine ausgezeichneten, stationären Server existieren müssen, aber sehr wohl existieren dürfen.

Aufgrund der genannten mannigfaltigen Vorteile bietet sich daher die Entwicklung einer möglichst infrastrukturlosen Multiple-Servents-Variante an. Als Vertreter einer solchen Variante werden im Folgenden P2P-Systeme dahingehend untersucht, ob die zugehörigen Ansätze und Algorithmen für das Management mobiler Prozesse adaptiert werden können.

3.6 Untersuchung der Eignung von P2P-Algorithmen für das Managementsystem

Reine Peer-to-Peer-Netze zeichnen sich dadurch aus, dass die einzelnen Teilnehmer als Servent agieren können, keine zentralen Server existieren und sich dementsprechend solche Netze dezentral selbst organisieren und verwalten. [Sch03] Im Abschnitt 3.5 wurde be-

reits evaluiert, dass eine infrastrukturlose Variante, bei der die einzelnen Teilnehmer bezüglich der Managementfunktionalität als Servent agieren, in diesem Kontext eine geeignete Architektur ist. Aufgrund der offensichtlichen Gemeinsamkeiten wird im Folgenden überprüft, ob die Algorithmen von Peer-to-Peer-Systemen auf mobile Ad-hoc-Netze anwendbar und übertragbar sind und ob eine derartige Adaption in Bezug auf die Ziele einer Managementkomponente sinnvoll ist.

Allgemeine Betrachtung von P2P-Systemen

Heutige Peer-to-Peer-Anwendungen ermöglichen Dienste wie File Sharing, Web Caching und Information Distribution, indem die Ressourcen von zehntausenden von Internetbenutzern genutzt werden. Die erste P2P Anwendung war *Napster*, die den Austausch von Musikdateien zwischen Benutzern im Internet ermöglichte, wobei hierbei zentralisierte Server zur Indexierung verwendet wurden, mit deren Hilfe Benutzer Dateien publizieren und suchen konnten. Der eigentliche Datenaustausch wurde hier allerdings direkt zwischen den jeweiligen Interessenten durchgeführt. Die später entwickelten *Gnutella* und *Freenet* sind Beispiele für File Sharing-Anwendungen, die völlig ohne zentralisierte Server auskommen. [CDK05]

P2P-Netze, die vollständig selbstorganisierend sind, werden auch *pure Peer-2-Peer-Netze* genannt, in denen oberhalb der IP-Schicht ein virtuelles *Overlay-Netzwerk* erzeugt wird. Unter einem *Overlay-Netzwerk* versteht man die Tatsache, dass das P2P-System ein auf einer konkreten Netzwerk-Topologie aufgebautes, übergeordnetes Netzwerk mit eigener unabhängiger Topologie darstellt. [HD05] Dieses Netzwerk besteht in den meisten Fällen lediglich aus den Servents und ihren TCP/IP-Verbindungen. Auf Basis dieses übergelagerten Netzwerkes werden Daten (z. B. Mp3-Dateien) direkt zwischen den einzelnen Teilnehmern ausgetauscht.

Vergleich zwischen Peer-to-Peer-Systemen und mobilen Ad-hoc-Netzen

Reine P2P-Netze sind genau wie MANETs selbstorganisierend, des Weiteren ist in beiden Systemen die Topologie dynamisch, in MANETs hauptsächlich aufgrund der vorhandenen Mobilität und in P2P-Netzen deshalb, weil die einzelnen Teilnehmer das Netz verlassen (z. B. durch Ausschalten des Rechners) oder neue das Netz betreten können. Damit neue Endgeräte die Möglichkeit haben, an dem jeweiligen Netzwerk teilzunehmen, ist in beiden Fällen gewisses Vorwissen unabdingbar. Bei P2P-Netzen muss die IP-Adresse wenigstens eines Teilnehmers (oder speziellen Servers) bekannt sein, um teilnehmen zu können, während in MANETs eine bestimmte Frequenz notwendig ist, über die die Teilnehmer ansprechbar sind.

Als Unterschied lässt sich aufführen, dass zum Beispiel der Datenaustausch in P2P-Netzen im Gegensatz zu mobilen Ad-hoc-Netzen zwischen den beiden beteiligten Interessenten direkt verläuft. Im Erstgenannten Netz wird die Route zwischen den Knoten

lediglich zum Aufbau einer Verbindung verwendet und nicht für den eigentlichen Datenaustausch, der hier direkt (zumindest aus der Perspektive des Overlay-Netzwerks) zwischen den beiden Interessenten abläuft. In MANETs muss jede Kommunikation über sämtliche Zwischenknoten stattfinden, was, da die einzelnen Knoten in MANETs potenziell mobil sind, die Aufrechterhaltung einer Verbindung deutlich erschwert. Des Weiteren herrscht in P2P-Netzen in der Regel keine Mobilität, d.h., die physikalische Position der einzelnen Knoten ist fest, während das in MANETs nicht der Fall ist. In Bezug auf das Routing funktionieren proaktive Routing-Algorithmen in P2P-Netzen im Gegensatz zu MANETs nicht, da gezeigt wurde (vgl. u.a. [JLH+99]), dass diese Art von Algorithmen nur mit weniger als 100 Knoten funktioniert und P2P-Netze im Gegensatz zu MANETs häufig eine deutlich größere Ausbreitung haben. Da die Peers in P2P-Netzen drahtgebunden interagieren, ist die Kommunikation und damit auch der allgemeine Verbindungsaufbau des Weiteren deutlich zuverlässiger als bei MANETs, bei denen Paketfehler aufgrund der unzuverlässigen drahtlosen Kommunikation deutlich wahrscheinlicher sind.

Weiterführend bleibt festzuhalten, dass die einzelnen Knoten in P2P-Netzen selbst meistens stationär und dementsprechend leistungsfähiger sind, wohingegen mobile Endgeräte diversen Restriktionen unterliegen (vgl. 2.2.4). [SGF02]

Fazit

Wie der vorangehende Vergleich gezeigt hat, existieren neben den offensichtlichen Gemeinsamkeiten, denen zufolge beide Arten von Systemen selbstorganisierend sind und dynamische Topologien besitzen, auch einige frappierende Unterschiede sowohl im Aufbau und Erhalt von Verbindungen als auch in Bezug auf allgemeine Faktoren wie Zuverlässigkeit, Fehlerrate, physikalische Ausbreitung und die prinzipiell verwendbaren Routing-Algorithmen. Neben diesen Unterschieden sind aber auch die Ziele von P2P-Systemen anders, so werden heutige P2P-Systeme hauptsächlich verwendet, um unveränderliche Daten wie Bilder, Musik oder allgemein Informationen auszutauschen, während in Ad-hoc-Netzen zumeist die Kommunikation und Kooperation zwischen den Nutzern im Vordergrund steht. Die Interaktionen in MANETs sind folglich eher benutzer- als datenzentriert. [SGF02]

Trotz der Gemeinsamkeiten zwischen P2P-Systemen und MANETs bietet es sich aufgrund der genannten Unterschiede und vor allem vor dem Hintergrund der divergierenden Ziele nicht an, bestehende P2P-Algorithmen für die Entwicklung der Managementkomponente zu adaptieren, da das Management eher gezielte Interaktionen zwischen Nutzern auf Basis veränderlicher Daten (Prozesse) und keine allgemeine Verteilung von unveränderlichen Daten und Inhalten zum Ziel hat. Dies soll nicht bedeuten, dass die allgemeinen P2P-Prinzipien nicht für das Management sinnvoll sind, sondern lediglich dass eine Adaption bestehender Algorithmen sich aufgrund genannter Unterschiede im Rahmen dieser Arbeit nicht anbietet.

3.7 Zusammenfassung

In diesem Kapitel wurde zunächst verdeutlicht, was allgemein unter dem Begriff „Management“ verstanden wird. Im Anschluss an diese Definition wurde das Workflow-Management aufgrund diverser Gemeinsamkeiten wie Verteilung, Computerunterstützung und Ausführung von Prozessen in Hinblick auf die für das Management mobiler Prozesse notwendige Funktionalität betrachtet. Mithilfe der gewonnenen Erkenntnisse wurde unter Berücksichtigung des aktuellen Forschungsstandes auf dem Gebiet des Managements im Bereich der mobilen Prozesse sowie der noch offenen Probleme definiert welche Aspekte (namentlich Recovery, Logging und Monitoring) für das Management mobiler Prozesse wichtig sind und noch adäquat umgesetzt werden müssen. Des Weiteren wurde die Umsetzung dieser Funktionen in anderen Gebieten untersucht, wobei sich herausgestellt hat, dass viele der Ansätze brauchbar, aber aufgrund der Mobilität und den hieraus resultierenden Restriktionen unter Einbeziehung der verwendeten Architektur angepasst werden müssen.

Als geeignete Architektur für die Managementkomponente hat sich hierbei eine infrastrukturlose Variante, bei der die einzelnen Teilnehmer im Netzwerk zusätzlich die Managementfunktionalität kooperativ bereitstellen, als vorteilhafter als eine infrastrukturbasierte Variante erwiesen. Da dieser Ansatz sehr stark den Prinzipien von P2P-Systemen ähnelt, wurde darüber hinaus untersucht, inwiefern derartige Algorithmen für mobile Prozesse anwendbar sind, wobei sich gezeigt hat, dass eine Adaption nicht sinnvoll ist.

Im folgenden Kapitel wird die Umsetzung der Managementkomponente im Detail besprochen, wobei die Erkenntnisse aus diesem Kapitel und insbesondere die als brauchbar identifizierten (aber anpassungsbedürftigen) Ansätze aus anderen Gebieten als Grundlage dienen.

4 Konzept zum Management mobiler Prozesse

Dieses Kapitel beschreibt das im Rahmen dieser Arbeit entwickelte Konzept zur Umsetzung einer Managementkomponente für mobile Prozesse. Zur Umsetzung der Managementkomponente und ihrer inhärenten Funktionalität wurden vornehmlich bereits bekannte, in anderen Gebieten etablierte Mechanismen und Methoden in adaptierter Form verwendet. Adaptiert bedeutet in diesem Kontext, dass die Methoden und Algorithmen an die Architektur und die allgemeinen Gegebenheiten in mobilen Systemen unter Berücksichtigung der systemspezifischen Beschränkungen und Restriktionen und den im vorherigen Kapitel ermittelten Anforderungen an eine Managementkomponente (vgl. 3.3), angepasst wurden.

Als Kernfunktionen des Managements für mobile Prozesse wurden im Kapitel 3.2 Logging, Monitoring und Recovery identifiziert, beschrieben und voneinander abgegrenzt. Hierbei hat sich herausgestellt, dass die einzelnen Teilbereiche eines solchen Managements starke Abhängigkeiten zueinander besitzen: So ist das Aufzeichnen von Logdateien beispielsweise eine wichtige Voraussetzung, um Fehler im Rahmen der Recovery (speziell Backward Recovery) behandeln zu können und des Weiteren auch wichtig, um adäquat Monitoring zu betreiben. Damit Fehler behoben werden können, müssen diese erst erkannt werden, folglich ist Monitoring wiederum eine wichtige Voraussetzung, um Recovery durchzuführen. Aufgrund der genannten Abhängigkeiten wird deutlich, dass die einzelnen Funktionen kaum isoliert umgesetzt werden können, weshalb innerhalb des Konzeptes keine strikte Trennung zwischen diesen Funktionen angestrebt wird.

Um zu verdeutlichen, wie die Interaktionen zwischen den Endgeräten im Hinblick auf das Management aussehen und wie die Managementkomponente selbst aufgebaut ist, wird der Ablauf zunächst allgemein, also in grober Granularität dargestellt und die einzelnen Aspekte und Mechanismen werden erst später im Detail besprochen.

4.1 Konzeptionelle Entscheidungen

Bevor nun die eigentliche Managementkomponente vorgestellt wird, wird in diesem Abschnitt ein kurzer Überblick darüber gegeben, welche konzeptionellen Entscheidungen bei der Entwicklung der Managementkomponente getroffen und aus welchen Bereichen bestehende Ansätze in adaptierter Form übernommen wurden.

4.1.1 Architektur

Aufgrund der im Abschnitt 3.5 verdeutlichten mannigfaltigen Vorteile einer infrastrukturlosen Variante gegenüber einer infrastrukturbasierten wird als zu Grunde liegende Architektur der Managementkomponente eine *Multiple-Servents-Variante* (vgl. 3.5.2), bei der mehrere Teilnehmer im Netzwerk das Management zusätzlich kooperativ bereitstellen, verwendet. Auf welche Art und Weise das Management eines Prozesses durchgeführt wird, wird über ein spezifisches Dokument, das so genannte *Steuerungsdokument* festgelegt, das genau wie die *Logdatei* in der etwaige Loginformationen zwischengespeichert werden, im Rahmen des Managements mit dem zugehörigen Prozess migriert. Die

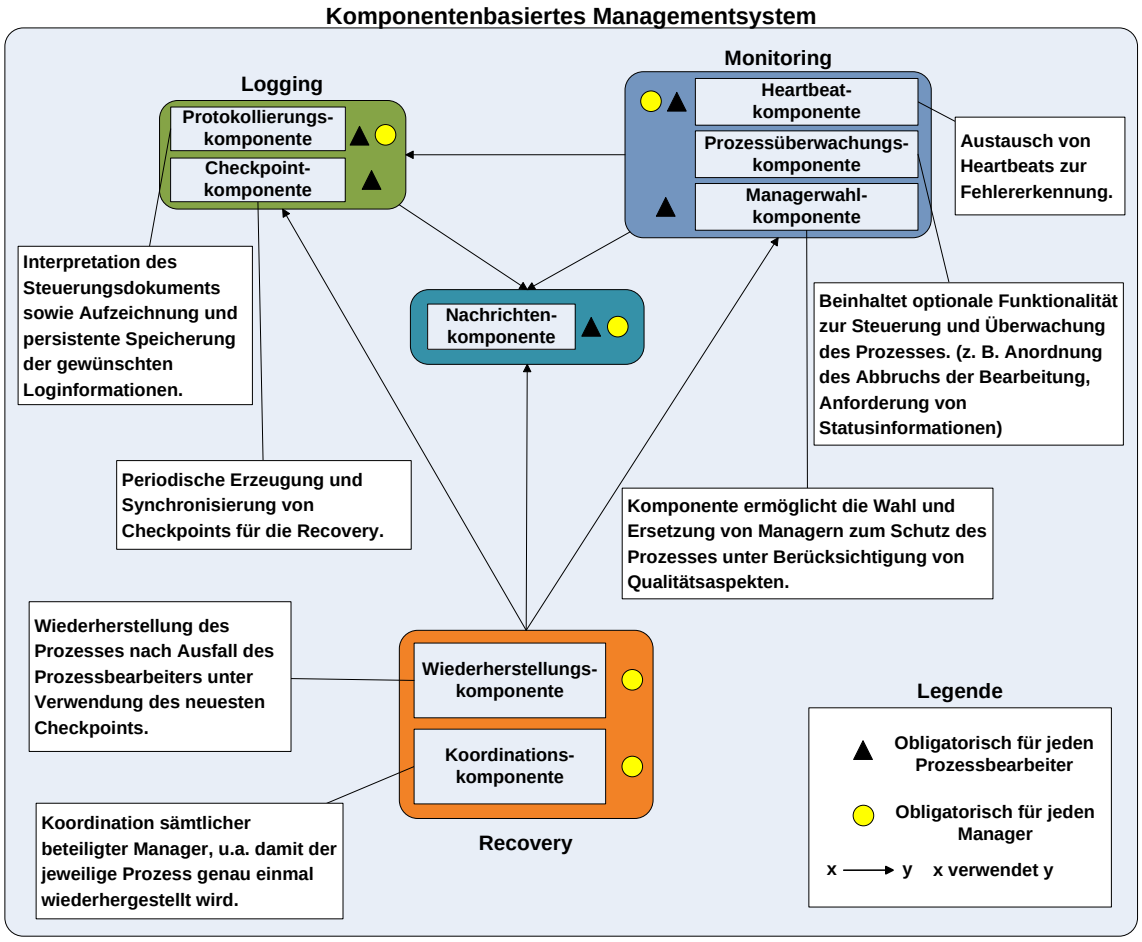

Abbildung 4.1: Architektur der Managementkomponente

Interaktionen zwischen den Endgeräten verlaufen beim Management auf Basis spezifischer Rollen, die festlegen, welches Endgerät welche Aktion in welcher Situation durchzuführen hat und die dynamisch, d.h. während der Laufzeit des Prozesses, unter Berücksichtigung von Qualitätsaspekten vergeben werden, da a priori in der Regel nicht bekannt ist, welche Endgeräte zu welchem Zeitpunkt verfügbar sind. Im Hinblick auf das zu Grunde liegende Netz wird von einem MANET (vgl. 2.2.7) ausgegangen, bei dem die Knotenanzahl variabel und unbekannt ist und in dem über potenziell unterschied-

liche Kommunikationstechnologien drahtlos per Nachrichtenaustausch interagiert und kooperiert wird. Die Hauptgründe für die Verwendung einer solchen Architektur sind die sehr gute Skalierbarkeit und die hohe Fehlertoleranz. Außerdem ermöglicht eine auf diese Art und Weise umgesetzte Managementkomponente den Einsatz in Gebieten, in denen keine Server aufgestellt werden können. Es handelt sich darüber hinaus auch um die kostenärmere Variante (vgl. Abschnitt 3.5.3).

Abbildung 4.1 verdeutlicht den Aufbau der Managementkomponente, das Zusammenspiel der einzelnen identifizierten Teilbereiche und veranschaulicht, welche Teilkomponenten für die Rollen des Bearbeiters eines Prozesses und eines Managers, der zum Schutz des Prozesses gewählt werden kann, essenziell sind (vgl. 4.2).

4.1.2 Recovery

Die Wiederherstellung von Prozessen und der dazu gehörenden Informationen im Rahmen der Recovery wird konzeptionell in der Managementkomponente über eine Rückwärtswiederherstellung (*Backward Recovery*) nach den im Abschnitt 3.4.1 beschriebenen Prinzipien geregelt. Hierbei werden zur Absicherung der Prozessbearbeitung periodisch Kontrollpunkte erstellt und extern, d.h. auf einem oder mehreren Endgeräten, die dynamisch ausgewählt werden, zwischengespeichert, um die Prozessbearbeitung vor Knotenausfällen und ihren Implikationen zu schützen. Da in einem infrastrukturlosen mobilen System nicht vorhergesagt werden kann, welche Endgeräte zu welchem Zeitpunkt im Netzwerk vorhanden sind und aufgrund der aus der Mobilität resultierenden Fehleranfälligkeit eine persistente Speicherung von Informationen (bzw. in diesem Kontext der Kontrollpunkte) in der Regel nicht hundertprozentig sichergestellt werden kann, ist dieser Ansatz charakteristisch als „best-effort“-Ansatz zu verstehen. Die Wahrscheinlichkeit einer erfolgreichen Prozessausführung wird demnach durch die redundante Speicherung essenzieller Informationen erhöht, kann aber aufgrund der Restriktionen in mobilen Systemen nicht vollständig garantiert werden.

4.1.3 Monitoring

Der zentrale Aspekt des Monitorings innerhalb der Managementkomponente ist die Feststellung von Knotenausfällen. Im Abschnitt 3.4.3 wurde evaluiert, dass ein an das jeweilige System angepasster adaptiver Heartbeat-Algorithmus für die Ausfallerkennung prinzipiell geeignet ist. Für die Managementkomponente wird der im genannten Abschnitt kurz vorgestellte Algorithmus von *Elhadef et al.* [EAN07] als Grundlage verwendet, da dieser Algorithmus speziell für mobile Ad-hoc-Netze entwickelt wurde und die Gegebenheiten in dieser Netzklasse von den betrachteten Algorithmen am besten berücksichtigt. Da die Managementkomponente für die Kooperation über mobile Prozesse ausgelegt ist, wurde der Algorithmus in einigen Bereichen an dieses Konzept angepasst, um die spezifischen Anforderungen zu erfüllen. Neben der Ausfallerkennung beinhaltet das

Monitoring innerhalb der Managementkomponente u.a. die Möglichkeit, Informationen über den aktuellen Status eines Prozesses abzufragen. Für diesen Ansatz wird *passives Monitoring* (vgl. 3.4.3) verwendet, d.h., der jeweilige Interessent schickt eine Anfrage in Form eines Broadcasts, um die gewünschten Statusinformationen zu erhalten, da im Allgemeinen nicht vorhergesagt werden kann, zu welchem/en Zeitpunkt(en) ein solches Interesse besteht und daher *aktives Monitoring* in der Regel nicht geeignet ist.

4.1.4 Logging

Im Abschnitt 3.4.2 wurden gängige Anwendungsgebiete für das Logging vorgestellt, also die Aufzeichnung von Informationen über etwaige Aktivitäten und Geschehnisse innerhalb eines Systems. Hierbei wurde aufgrund der vielen Analogien zwischen dem Kooperationskonzept der mobilen Prozesse und der Bearbeitung von Geschäftsprozessen in Workflow-Managementsystemen der Aspekt des Loggings im Workflow-Management näher betrachtet. Im Rahmen dieser Betrachtung wurden Logeinträge identifiziert, die auch im Kontext der mobilen Prozesse sinnvoll sind bzw. sein können, wie beispielsweise das Aufzeichnen des Start- und Endes einer Aktivität, um, basierend auf diesen Informationen, die Dauer bestimmter Aktivitäten festzuhalten und für spätere Analysen zu verwenden. Bei der Festlegung, welche Informationen über bzw. mithilfe der Managementkomponente aufgezeichnet werden sollen, wurden die beim Workflow-Management identifizierten Logeinträge als Grundlage verwendet und um spezifische für das Management notwendige Felder bzw. Einträge erweitert. Der Ansatz selbst wurde hierbei möglichst generisch gehalten, um den Loggingprozess möglichst offen zu halten und damit prinzipiell jede Art von Information aufzeichnen zu können.

4.2 Rollen im Managementsystem

Bevor die einzelnen möglichen Interaktionen beschrieben werden, ist es zunächst notwendig zu verstehen, welche Rollen es innerhalb des Managementsystems gibt. Es werden hierbei drei Rollen unterschieden, von denen jede im Kontext der Ausführung mobiler Prozesse sowie des zugehörigen Managements unterschiedliche Aufgaben beinhaltet:

- **Prozessinitiator**: Als Prozessinitiator wird der Benutzer bezeichnet, der für einen mobilen Prozess sämtliche managementspezifischen Parameter festlegt. Seine rollenspezifischen Aufgaben bestehen darin zu spezifizieren, ob der mobile Prozess Managementfunktionalität in Anspruch nehmen soll und wenn ja, in welchem Umfang. Im Detail beschreibt er im so genannten *Steuerungsdokument*, welche Informationen in der *Logdatei* gespeichert werden sollen und wie viele Manager zur Überwachung der Prozessausführung akquiriert werden sollen, um im Falle eines Knotenausfalles korrigierend eingreifen zu können und damit eine korrekte Prozessausführung trotz etwaiger Fehler zu gewährleisten. Des Weiteren beinhaltet die

Rolle die Möglichkeit, den Bearbeitungsstatus des eigenen Prozesses zu ermitteln und bei Bedarf steuernd einzugreifen (z. B. den Prozess abzubrechen).

- **Prozessbearbeiter**: Als Prozessbearbeiter wird das Endgerät bezeichnet, das einen Prozess bzw. eine oder mehrere Aktivitäten eines Prozesses zu einem Zeitpunkt bearbeitet. Neben der Ausführung von Aktivitäten ist der Prozessbearbeiter dafür zuständig, die im Prozess (vom Prozessinitiator) spezifizierten Loginformationen (z. B. Bearbeitungsstatus, Kontrollpunkte für Recovery) zu erheben und, sofern spezifiziert, diese Informationen periodisch an einen (oder mehrere) Manager zu übermitteln.

- **Manager**: Ein Manager ist eine Rolle im Managementsystem, die die Überwachung von mobilen Prozessen zur Aufgabe hat. Fällt ein Prozessbearbeiter aus, so fällt ebenfalls jeder Prozess aus, der von dem Prozessbearbeiter bearbeitet wurde. Damit die Bearbeitung in einem solchen Fall nicht vollkommen zum Stehen kommt und im schlimmsten Fall erneut ganz von vorne durchgeführt werden muss, ist es notwendig, dass der aktuelle Bearbeitungsstatus in Form von Kontrollpunkten für die Ausführung extern (also auf anderen Endgeräten) gespeichert wird, um trotz des Ausfalles des Prozessbearbeiters die Prozessausführung des jeweiligen Prozesses vom möglichst jüngsten konsistenten Zustand aus weiterführen zu können. Die Speicherung wichtiger prozessrelevanter Informationen sowie die Überwachung des Prozessbearbeiters selbst im Sinne einer Ausfallerkennung sind folglich im Kern die vom Manager durchzuführenden Aufgaben.

Da es in einem MANET per Definition keine Infrastruktur und damit auch keine zentralen Server gibt, die z. B. die Rolle des Managers ausführen können, müssen folglich die Teilnehmer selbst sämtliche Rollen einnehmen. Hierbei bleibt festzuhalten, dass die Rollen prozessspezifisch sind, d.h., ein Endgerät kann für einen mobilen Prozess als Prozessinitiator fungieren, für einen anderen Prozess als Manager und wiederum für einen anderen Prozess als Prozessbearbeiter. Des Weiteren sind die Rollen (abgesehen vom Prozessinitiator) nicht als exklusiv zu verstehen, d.h. ein Prozess kann mehrere Prozessbearbeiter/Manager zu einem Zeitpunkt haben und ein Endgerät kann auch für mehrere Prozesse als Prozessbearbeiter/Manager gleichzeitig fungieren. Darüber hinaus können die Rollen sich (abgesehen von der Prozessinitiator-Rolle) im zeitlichen Verlauf ändern. So ist ein Endgerät, das einen Prozess aus einem beliebigen Grund nicht weiter bearbeiten kann und deshalb diesen zu einem leistungsfähigeren Endgerät migrieren lässt, nach Abschluss der Migration für diesen Prozess rollenspezifisch kein Prozessbearbeiter mehr.

4.3 Ablauf des Managements

In diesem Abschnitt wird der allgemeine Ablauf des Managements mobiler Prozesse sowie der zugehörigen rollenspezifischen Interaktionen beschrieben. Jedes Endgerät (bzw.

der zugehörige Benutzer) im mobilen System hat prinzipiell die Möglichkeit, eigene Prozesse zu spezifizieren und zur Bearbeitung frei zu geben, sodass zu einem Zeitpunkt eine Vielzahl an mobilen Prozessen nebenläufig bzw. parallel und dezentral im System ausgeführt werden kann. Da die mobilen Prozesse das zentrale Element sind, das überwacht werden soll bzw. von dem Informationen aufgezeichnet werden sollen und im Normalfall keine Interaktionen zwischen verschiedenen mobilen Prozessen bestehen, bietet es sich folglich an, sämtliche Interaktionen anhand des allgemeinen Lebenszyklus eines mobilen Prozesses zu beschreiben.

4.3.1 Spezifikation des Managementgrades

Abbildung 4.2 verdeutlicht grafisch den im Folgenden beschriebenen Lebenszyklus anhand der einzelnen charakteristischen Phasen zur Veranschaulichung. Der erste Schritt

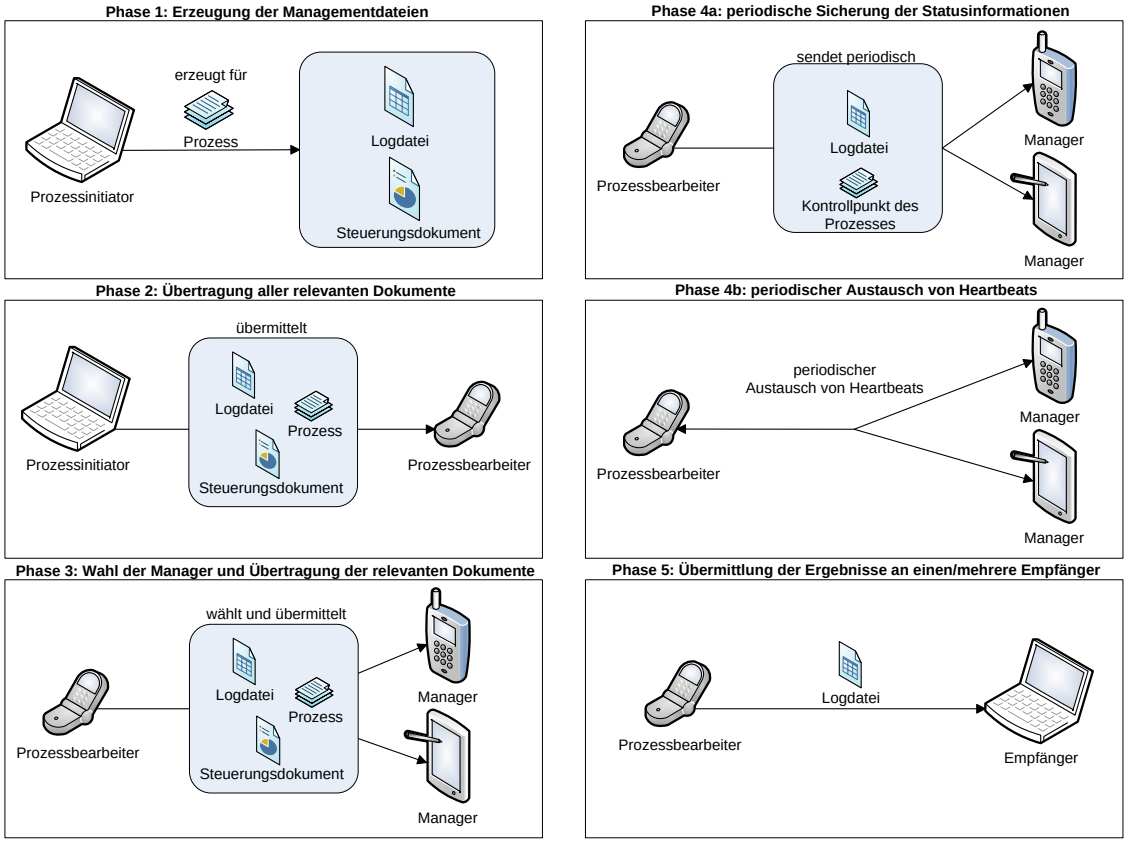

Abbildung 4.2: Ablauf des Managements eines Prozesses

im Lebenszyklus eines Prozesses besteht in der Spezifikation der Aktivitäten des Prozesses. Es wird folglich zunächst mit einer Definitionssprache festgelegt, welche Aktivitäten in welcher Reihenfolge und unter welchen Bedingungen durchgeführt werden. Im Rahmen des Managements ist hierbei die Aufgabe des Prozessinitiators festzulegen, ob Managementfunktionalität bezüglich des Prozesses genutzt werden soll und in welchem Umfang. Im Detail erzeugt der Prozessinitiator ein Steuerungsdokument, in dem festge-

halten wird, welche Aspekte des Managements genau genutzt werden sollen. Es ist hierbei nicht erforderlich, dass der Prozessinitiator den Prozess selbst spezifiziert hat, wobei dies sicherlich häufig der Fall ist bzw. sein wird, da detailliertes Wissen über den Prozess notwendig ist, um festlegen zu können, was genau aufgezeichnet werden soll und wie das Management im Detail auszusehen hat. Im Steuerungsdokument selbst wird unter anderem festgelegt, ob die Ausführung des Prozesses von Managern auf Basis extern gespeicherter Kontrollpunkte (also im Rahmen einer Backward Recovery) abgesichert werden soll oder ob auf diese Maßnahme verzichtet wird. Der Prozessinitiator legt demgemäß fest, wie viele Manager hierzu verwendet werden sollen. Des Weiteren wird festgelegt, welche ausführungsspezifischen Informationen im Rahmen des Loggings aufgezeichnet werden sollen. Neben der Aufzeichnung darüber, welche Endgeräte (Prozessbearbeiter/Manager) an der Prozessbearbeitung beteiligt waren, können beispielsweise Informationen darüber aufgezeichnet werden, wie lange einzelne Aktivitäten gedauert haben, wie oft der Prozess im Rahmen einer Recovery wiederhergestellt werden musste und welche (Zwischen-)Ergebnisse produziert wurden. Außerdem besteht die Möglichkeit festzulegen, an wen die Ergebnisse und Loginformationen des Prozesses übermittelt werden sollen.

Nach Abschluss der Definition sämtlicher für den Prozessinitiator relevanter Aspekte erzeugt er, sofern Informationen aufzeichnet werden sollen, eine separate Logdatei, in der diese Informationen gespeichert werden (*Phase 1*). Falls der Prozessinitiator die Möglichkeit besitzt, die erste Aktivität im Prozess selbst zu bearbeiten, nimmt er neben der Prozessinitiator-Rolle die in der Regel temporäre Rolle des Prozessbearbeiters für den mobilen Prozess ein. Hat der Prozessinitiator keine Möglichkeit, die nächste Aktivität zu bearbeiten oder hat er bereits zu viele weitere Aufgaben, so lässt er den Prozess inklusive des Steuerungsdokuments und Logdatei zu einem Endgerät migrieren, das die nächste Aktivität bearbeiten kann und damit die Rolle des Prozessbearbeiters einnimmt (*Phase 2*). Der Prozessinitiator gibt folglich nach der Migration die Kontrolle über den Prozess vollständig ab, hat aber im Rahmen der Managementkomponente die Möglichkeit, Informationen über den derzeitigen Status der Prozessbearbeitung abzufragen oder sogar den Abbruch der Bearbeitung anzuordnen sowie diese Rechte auch anderen Endgeräten zu übertragen, indem er das Steuerungsdokument entsprechend spezifiziert.

4.3.2 Unterstützung der Prozessbearbeitung

Bevor der Prozessbearbeiter mit der Bearbeitung der nächsten anstehenden Aktivität beginnt, besteht seine Aufgabe zunächst darin, die im Steuerungsdokument spezifizierte Anzahl an Managern zu rekrutieren und den Prozess und seine zugehörigen Dokumente (Steuerungsdokument, Logdatei) an diese zu übermitteln, damit im Falle eines Ausfalles der Prozess wieder aufgesetzt werden kann. Bei der Auswahl der Manager können hierbei Qualitätsparameter (z. B. Batterielebensdauer, Rechenleistung) berücksichtigt werden (*Phase 3*). Sollten nicht genügend Manager zur Verfügung stehen, so lässt der Prozessbe-

arbeiter den Prozess zu einem anderen Endgerät migrieren.

Da die Manager bei einem Ausfall interagieren müssen (damit der Prozess beispielsweise nur einmal aufgesetzt wird), muss jedem Manager jeder andere bekannt sein, weshalb die Identifikatoren der aktuellen Manager im Steuerungsdokument gespeichert und bei Änderungen aktualisiert werden müssen. Damit bei einem Ausfall des Prozessbearbeiters die Bearbeitung nicht wieder von vorne beginnen muss und sämtliche Ergebnisse erhalten bleiben, übermittelt der Prozessbearbeiter an seine Manager periodisch den Prozess inklusive seiner aktuellen Zustandsinformationen sowie die aktuelle Logdatei (*Phase 4a*). Änderungen im Steuerungsdokument werden direkt per Nachricht propagiert. Zur Fehlererkennung sendet der Prozessbearbeiter darüber hinaus periodisch Heartbeat-Nachrichten an alle seine Manager. Da die Manager selbst auch ausfallen können und der Prozessbearbeiter in diesem Fall zum Schutz des Prozesses neue akquirieren sollte, senden die Manager darüber hinaus ebenfalls periodisch Heartbeat-Nachrichten an den Prozessbearbeiter (*Phase 4b*).

Während der Bearbeitung der anstehenden Aktivitäten ist der Prozessbearbeiter darüber hinaus dafür zuständig, sämtliche im Steuerungsdokument spezifizierten Informationen in die Logdatei zu schreiben. Sobald der Prozessbearbeiter die nächste anstehende Aktivität nicht mehr selber bearbeiten kann, sucht er sich ein dazu fähiges Endgerät und lässt den Prozess zu diesem Endgerät migrieren. Das alte Endgerät gibt damit die Rolle des Prozessbearbeiters an das neue Endgerät ab, gleichzeitig werden die Manager informiert und übernehmen nun die Aufgabe des Überwachers für den neuen Prozessbearbeiter, sofern sie noch als geeignet gelten, ansonsten kann dieser erneut nach oben genanntem Schema Manager wählen.

Die in diesem Abschnitt beschriebenen Schritte (*Phase 2-4*) werden so lange wiederholt, bis der komplette Prozess bearbeitet ist. Nach Abschluss des Prozesses werden die Manager über den Umstand informiert und die Ergebnisse und Logdaten, sofern erwünscht, an einen oder mehrere im Steuerungsdokument spezifizierte Empfänger gesendet (*Phase 5*). Wie diese Übermittlung im Detail aussieht, wird im Abschnitt 4.6 näher beschrieben.

4.4 Fehlerbehandlung im Managementsystem

Im vorherigen Abschnitt wurde der Ablauf des Managements anhand des Lebenszyklus eines Prozesses verdeutlicht. Hierbei wurde zunächst nicht auf spezifische Fehler, die bei der Bearbeitung auftreten können, eingegangen, d.h. es wurde, um einen Überblick über die allgemeinen Interaktionen, die Rollenvergabe und den Ablauf zu bekommen, der Optimalfall betrachtet. In diesem Abschnitt wird nun auf mögliche Fehler und Ausnahmesituationen eingegangen, um zu verdeutlichen, wie die Managementkomponente diese Situationen bewältigt.

4.4.1 Knotenausfall

Eine der Hauptaufgaben der Managementkomponente ist es, im Rahmen der Recovery, adäquat auf Knotenausfälle zu reagieren. Entsprechend spielen im Kontext des Managements ledigliche Ausfälle von Knoten, die zum Zeitpunkt des Ausfalles die Managementrolle Prozessbearbeiter und/oder Manager haben, eine Rolle, da Ausfälle anderer Knoten in der Regel keine negativen Auswirkungen auf die Prozessbearbeitung haben. Im Folgenden wird daher detailliert beschrieben, wie auf Ausfälle eines Prozessbearbeiters und/oder eines oder mehrerer Manager reagiert wird.

Ausfall eines Prozessbearbeiters

Beim Ausfall des Prozessbearbeiters lassen sich im Hinblick auf das Management grob drei unterschiedliche Szenarien unterscheiden:

- **Der Prozessbearbeiter hat keinen Manager:** Sofern vom Prozessinitiator festgelegt wurde, dass kein Manager und damit keine Absicherung der Prozessbearbeitung für den jeweiligen Prozess stattfinden soll, führt der Ausfall des Prozessbearbeiters unweigerlich zum Verlust des aktuellen Fortschritts des jeweiligen Prozesses inklusive der aufgezeichneten Logdaten und des zugehörigen Steuerungsdokuments.

- **Der Prozessbearbeiter hat einen Manager:** Da der Prozessbearbeiter im Rahmen des Monitorings Heartbeats mit seinem Manager austauscht, impliziert der Ausfall des Prozessbearbeiters, dass dieser keine Heartbeat-Nachrichten mehr versendet. Nach Ablauf des über den Heartbeat-Algorithmus berechneten Timeout-Intervalls für den nächsten Heartbeat beginnt der Manager direkt mit der Wiederherstellung des Prozesses, wobei der aktuelle Kontrollpunkt verwendet wird, damit die Bearbeitung nicht komplett von vorne durchgeführt werden muss. Im Anschluss an die Wiederherstellung lässt der Manager den Prozess inklusive der managementspezifischen Dateien (Steuerungsdokument, Logdatei) zu einem geeigneten Endgerät migrieren, woraufhin die Bearbeitung von diesem Endgerät bei gleich bleibendem Manager fortgesetzt wird. Fallen Manager und Prozessbearbeiter gleichzeitig aus, so gilt das zuvor beschriebene Szenario.

- **Der Prozessbearbeiter hat zwei oder mehr Manager:** Wie bereits im vorherigen Szenario beschrieben, führt der Ausfall des Prozessbearbeiters dazu, dass keine Heartbeats mehr an die Manager versendet werden. Dies impliziert wiederum, dass der Manager mit dem kürzesten Timeout-Intervall (also in der Regel derjenige, der am wenigsten Hops vom Prozessbearbeiter entfernt war) den Ausfall zuerst bemerkt. Damit ein Prozess nicht von jedem der Manager neu aufgesetzt wird, ist daher Koordination zwischen den Managern notwendig, um zu bestimmen, welcher der Manager diese Aufgabe übernimmt. Abbildung 4.3 verdeutlicht die so genannte *Recovery-Koordination*. Sobald das jeweilige Timeout-Intervall abge-

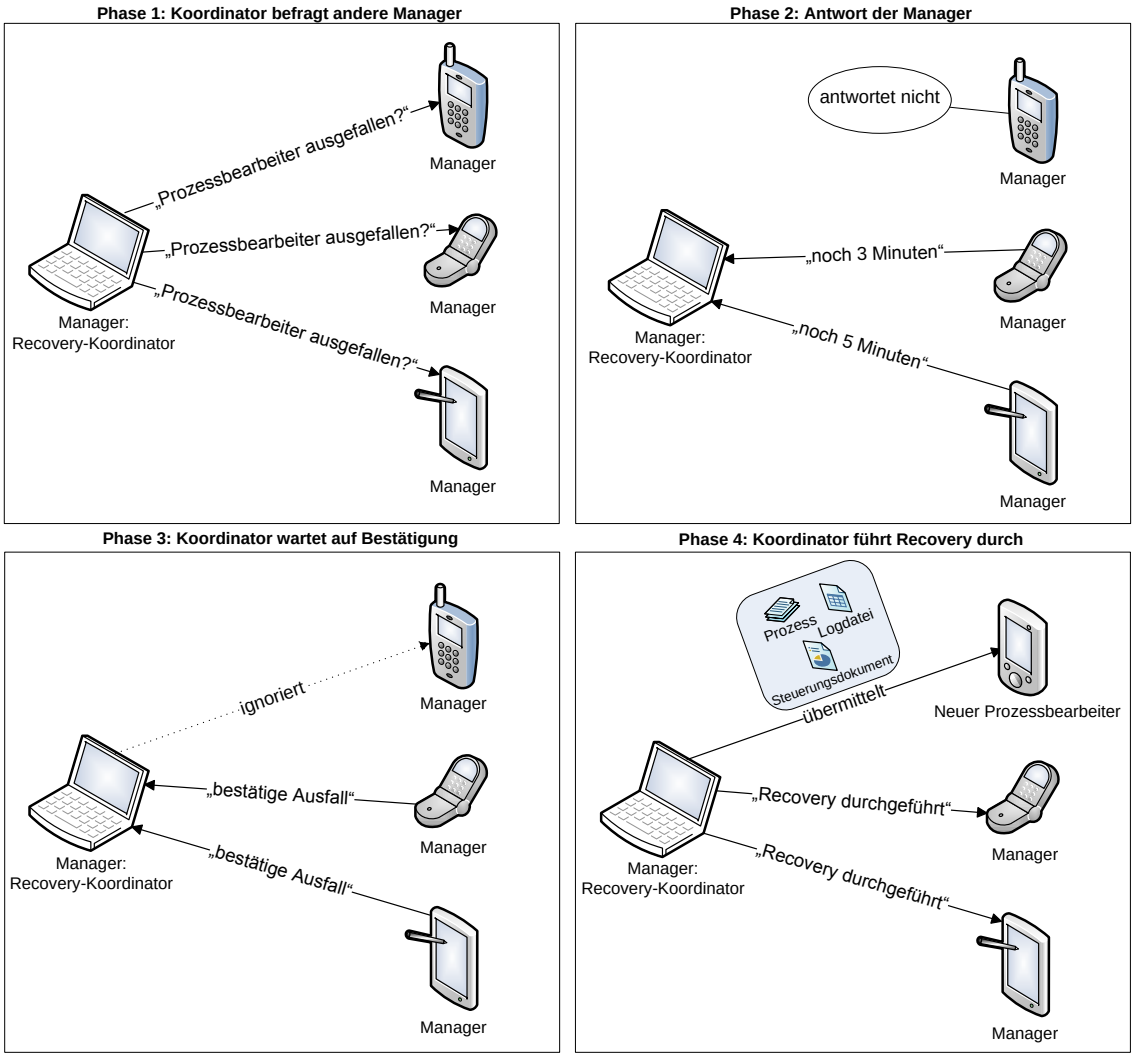

Abbildung 4.3: Ablauf der Recovery-Koordination

laufen ist, kontaktiert ein Manager zunächst sämtliche anderen prozessspezifischen Manager per Broadcast (die Identifikatoren der Manager findet er im Steuerungsdokument), d.h., er fragt an, ob die anderen den Ausfall ebenfalls bemerkt haben und wartet ein festgelegtes Zeitintervall auf die Antworten. Die Nachricht selbst beinhaltet hierbei die ID des Prozesses, um den es geht sowie den Identifikator des anfragenden Managers, der im Rahmen der Wiederherstellung des Prozesses als *Recovery-Koordinator* fungiert (*Phase 1*). Jeder Manager, der diese Nachricht erhält, sendet eine Antwortnachricht, die diejenige Zeit beinhaltet, die der Manager noch wartet, bis er den Prozessbearbeiter ebenfalls des Ausfalles verdächtigt (also das verbleibende Timeout-Intervall) (*Phase 2*). Nach Ablauf dieses Zeitintervalls schickt der jeweilige Manager eine Bestätigung, dass er den Prozessbearbeiter jetzt ebenfalls verdächtigt. Alle Manager, die nicht innerhalb des spezifischen Zeitintervalls auf die Anfrage des Koordinators antworten, werden vom Koordinator des Ausfalls verdächtigt und für die weitere Entscheidungsfindung ignoriert (*Phase 3*).

Der Recovery-Koordinator sammelt für sämtliche antwortenden Manager die Bestätigungsnachrichten und wartet, sofern ein Manager sendet, dass sein Timeout-Intervall noch nicht abgelaufen ist, die in der Nachricht enthaltene Zeit zusätzlich. Bekommt er nach Ablauf dieses Zeitintervalls sowie einer kurzen Sicherheitsmarge keine Antwort, so wird dieser Manager ebenfalls des Ausfalls verdächtigt und für die weitere Entscheidungsfindung ignoriert. Sobald der Koordinator von jedem (nicht ignoriertem) Manager eine Bestätigung erhalten hat, dass der Prozessbearbeiter des Ausfalls verdächtigt wird, beginnt dieser mit der Wiederherstellung des Prozesses und teilt sämtlichen anderen Managern mit, dass der Prozess jetzt neu gestartet wird (*Phase 4*). Nach Abschluss der Wiederherstellungsprozedur lässt der Koordinator den Prozess zu einem geeigneten Endgerät migrieren (inklusive der managementspezifischen Dateien) und die Prozessbearbeitung wird fortgesetzt, d.h., der neue Prozessbearbeiter übernimmt die alten Manager und tauscht nun mit ihnen Heartbeat-Nachrichten aus, bzw. wählt neue für diejenigen, die er nicht erreichen kann. Neben dem genannten Ablauf existieren noch einige zusätzliche Situationen, die im Folgenden betrachtet werden.

- *Temporärer Ausfall des Prozessbearbeiters / Fehlalarm:* Sollte während der Koordination zwischen den Managern eine neue Heartbeat-Nachricht eintreffen, so sendet derjenige Manager, der diesen Heartbeat erhält, eine Recovery-Abbruch-Nachricht, die die ID des Prozesses beinhaltet, an alle anderen Manager. Diese Abbruch-Nachricht führt dazu, dass die Recovery-Koordination beendet wird und jeder Manager, der noch keine Heartbeat-Nachricht bekommen hat, sein letztes berechnetes Timeout-Intervall erneut wartet. Bekommt er innerhalb des Zeitintervalls eine aktuelle Heartbeat-Nachricht, so wird normal weiterverfahren, ansonsten startet der jeweilige Manager nach Ablauf eine neue Recovery-Koordination. Eine Koordination kann auf diese Weise solange abgebrochen werden, bis der Koordinator die Nachricht sendet, dass der Prozess neu gestartet wurde. Sollte eine Heartbeat-Nachricht später eintreffen, so wird der alte Prozessbearbeiter über den Umstand informiert, damit er seine Arbeit abbricht und der Prozess damit nicht doppelt bearbeitet wird.
- *Mehrere Manager starten eine Recovery-Koordination (nahezu) gleichzeitig:* Sollten zwei (oder mehr) Manager in etwa gleiche Timeout-Intervalle haben, so kann es passieren, dass beide eine Recovery-Koordination als Koordinator starten. In diesem Fall entscheidet die Reihenfolge der Identifikatoren der Manager im Steuerungsdokument über das weitere Verfahren. Der Manager, dessen Identifikator vor dem Identifikator des anderen aufgeführt wird, hat höhere Priorität. Sobald ein Manager eine Koordinations-Nachricht mit höherer Priorität bekommt, bricht er die Koordination mit geringerer Priorität ab und nimmt fortan an der mit höherer Priorität teil.
- *Ausfall des Recovery-Koordinators:* Damit im Falle des Ausfalles des Recovery-

Koordinators sämtliche anderen Manager nicht beliebig lange auf weitere Anweisungen warten, ist ein zusätzlicher Ausfallerkennungs-Mechanismus für den Recovery-Koordinator notwendig. Eine Möglichkeit bestände darin, den Recovery-Koordinator ebenfalls mithilfe des Heartbeat-Algorithmus zu schützen. Dieser Ansatz würde aber zum einen zu weiterem Aufwand führen, da speziell für jede Recovery-Koordination Heartbeats ausgehandelt und ausgetauscht werden müssten, darüber hinaus könnten zum anderen die „Überwacher“ ebenfalls ausfallen und/oder müssten sich einigen, wer nun der neue Recovery-Koordinator wird, sofern ein Ausfall aufgedeckt wird. Aus den genannten Gründen wird daher ein aufwandsärmerer Ansatz priorisiert, der im Folgenden näher beschrieben wird:

Zur vereinfachten Ausfallerkennung des Koordinators verwendet jeder Manager, der an einer Recovery-Koordination teilnimmt (abgesehen vom Recovery-Koordinator) ein festgelegtes Zeitintervall (*Koordinatordeadline*), das zurückgesetzt wird, sobald eine neue Nachricht des Koordinators ankommt. Sofern das Zeitintervall abläuft, verdächtigt der jeweilige Manager den Recovery-Koordinator des Ausfalls und startet eine neue Koordination, in der er selbst die Rolle des Recovery-Koordinators übernimmt. Damit auch Manager mit geringerer Priorität als neuer Recovery-Koordinator auftreten können, enthält jede Nachricht der Recovery-Koordination eine so genannte *RecoveryID*, die bei jedem neuen Recovery-Koordinationsversuch inkrementiert wird. Treffen bei einem Knoten zwei unterschiedliche Recovery-Anfragen von unterschiedlichen Koordinatoren ein, so entscheidet zunächst die RecoveryID darüber, welche Koordination mit welchem Koordinator weiterverfolgt wird. Bei gleicher RecoveryID entscheidet die Priorität der Manager, die sich aus der Reihenfolge der Identifikatoren im Steuerungsdokument ergibt. Um zu vermeiden, dass sämtliche Manager nahezu zeitgleich eigene Koordinationen starten, wird die Koordinatordeadline bei jedem Manager unterschiedlich festgelegt. Der Manager, dessen Identifikator im Steuerungsdokument an letzter Stelle aufgeführt wird, bekommt die kürzeste Wartezeit. Die Koordinatordeadline des Endgeräts, dessen Identifikator im Steuerungsdokument an vorletzter Stelle aufgeführt wird, ergibt sich aus der Addition der Koordinatordeadline des letzten Endgeräts im Steuerungsdokument und einem zusätzlichen festgelegten Zeitintervall. Die Koordinatordeadline des drittletzten Endgeräts ergibt sich wiederum aus der Addition der Koordinatordeadline des vorletzten Endgeräts und dem festgelegten zusätzlichen Zeitintervall und so weiter.

Ausfall eines Managers/mehrerer Manager

Da die Manager ebenfalls an den Prozessbearbeiter periodisch Heartbeat-Nachrichten versenden, führt das Ausbleiben einer solchen Nachricht nach Ablauf des Timeout-Intervalls

dazu, dass der Prozessbearbeiter den Manager des Ausfalls verdächtigt. In diesem Fall wählt der Prozessbearbeiter einen neuen Manager, aktualisiert den zugehörigen Eintrag im Steuerungsdokument und sendet die aktualisierte Version an alle Manager. Sollte der Manager nach Abschluss der Wahl wieder auftauchen, also lediglich temporär ausgefallen oder außer Kommunikationsreichweite gewesen sein, so teilt der Prozessbearbeiter dem Manager mit, dass er nicht mehr zuständig ist. Sofern zu diesem Zeitpunkt noch kein neuer Manager gewählt wurde, wird die Arbeit mit dem alten Manager fortgesetzt.

4.4.2 Nicht behandelte Fehler

Neben dem Ausfall von Knoten können z. B. auch Fehler bei der Ausführung von Aktivitäten auftreten. Diese Fehlerart wird vom Managementsystem bewusst nicht behandelt, sondern sollte bei der Definition der Prozesslogik vom Erzeuger des Prozesses festgelegt werden (z. B. über Schleifen und/ oder Verzweigungen). Des Weiteren wird von byzantinischen Fehlern abstrahiert, d.h. mit anderen Worten: Knoten „lügen“ nicht und senden keine böswilligen/zufälligen Nachrichten.

4.5 Implikationen der Manageranzahl

Eine der zentralen Entscheidungen, die der Prozessinitiator für seinen Prozess treffen muss, ist, wie viele Manager zur Absicherung seines Prozesses verwendet werden sollen. Um diese Frage zu beantworten, ist es notwendig zu verstehen, welche Auswirkungen eine hohe oder niedrige Anzahl bzw. der komplette Verzicht auf Manager beinhalten.

Die Hauptaufgabe der Manager ist die Überwachung des Prozessbearbeiters mithilfe von Heartbeat-Nachrichten, um auf einen potenziell auftretenden Ausfall des Prozessbearbeiters adäquat reagieren zu können, also den Prozess von einem aktuellen Kontrollpunkt aus neu starten zu können. Hier muss berücksichtigt werden, dass sich nicht unterscheiden lässt, ob ein Knoten tatsächlich ausgefallen oder nur besonders langsam ist (vgl. 3.4.3). Des Weiteren besteht die Möglichkeit, dass aufgrund der Mobilität für eine bestimmte Zeit keine Route zwischen Manager und Prozessbearbeiter besteht, sodass aufgrund ausbleibender Heartbeat-Nachrichten Recovery-Maßnahmen eingeleitet werden, obwohl der Prozessbearbeiter nicht ausgefallen ist. Da diese Fälle nicht unterschieden werden können, kann es passieren, dass ein Manager einen Prozess neu startet, obwohl der Prozessbearbeiter noch funktionstüchtig ist, d.h., eine doppelte Ausführung kann bei der Verwendung von Managern nicht gänzlich ausgeschlossen werden. Verwendet man hingegen keinen Manager, so führt der Ausfall des Prozessbearbeiters zum Ausfall des Prozesses und damit zum Verlust sämtlicher Informationen, die notwendig sind, um den Prozess von einem aktuellen Zustand aus neu starten zu können, was in den meisten Fällen aufgrund der hohen Fehleranfälligkeit in mobilen Systemen ebenfalls keine Lösung ist. Bei der Anzahl der Manager ist demnach Folgendes zu beachten: Je mehr Manager gewählt werden, desto geringer ist die Gefahr, dass der Prozess komplett ausfällt, aller-

dings erhöht sich mit steigender Anzahl an Managern das Nachrichtenaufkommen. So werden z. B. zusätzliche Nachrichten benötigt, um Manager zu wählen bzw. im Falle eines Ausfalles zu ersetzen, um Heartbeat-Nachrichten auszutauschen und für die Koordination zwischen den Managern. Da das Versenden von Nachrichten energieintensiv ist und mobilen Endgeräten Energie zumeist nur in begrenztem Umfang zur Verfügung steht, sollte dies folglich bei der Festlegung der Manageranzahl berücksichtigt werden. Abgesehen davon steigt mit der Anzahl der Manager ebenfalls die Gefahr, dass Prozesse doppelt oder sogar mehrfach zur gleichen Zeit ausgeführt werden.

Im Großen und Ganzen muss der Prozessinitiator also abwägen, welche der genannten Aspekte (mehrfache Ausführung des Prozesses, Energieintensität, potenzieller Verlust des Prozesses) am wichtigsten für ihn sind und dementsprechend gewichten, wobei natürlich die Ausführungsumgebung und spezifische Randbedingungen (sofern möglich) mit berücksichtigt werden können und sollten. Ist die Gefahr eines Ausfalles in einem bestimmten mobilen System aufgrund bestimmter Randbedingungen sehr klein, so ist der Schutz des Prozesses mithilfe von wenigen oder sogar ohne Manager ggf. ausreichend.

4.6 Prozessüberwachung

Im Rahmen des Managements hat der Prozessinitiator die Möglichkeit, im Steuerungsdokument festzulegen, welche Endgeräte den Status der Prozessbearbeitung abfragen dürfen und welche Endgeräte die Möglichkeit haben, die Prozessbearbeitung zu stoppen, also den Abbruch anzuordnen. In diesem Abschnitt wird beschrieben, wie eine derartige Prozessüberwachung im Rahmen des Managements gehandhabt wird.

4.6.1 Steuerungsnachricht- und Antwortübermittlung

Da MANETs per Definition mobil sind, d.h., Routen sich ständig ändern und in der Regel nicht jeder Teilnehmer mit jedem direkt kommunizieren kann, ist das Auffinden des Prozessbearbeiters keineswegs trivial. So kann aufgrund dieser Dynamik nicht vorhergesagt werden, wo sich ein Prozess zu einem Zeitpunkt befindet und ob überhaupt eine Route zwischen dem Statusanfragenden und dem Prozessbearbeiter existiert. Der folgende Ansatz besteht aus zwei unterschiedlichen Phasen, die das Auffinden des Prozessbearbeiters und damit die Übermittlung einer *Steuerungsnachricht* ermöglichen. Eine *Steuerungsnachricht* ist hierbei eine Nachricht, die für einen bestimmten Prozessbearbeiter bestimmt ist und spezifische Anweisungen wie z. B. das Senden des aktuellen Status der Bearbeitung oder die Anordnung des Abbruches der Prozessbearbeitung beinhaltet. Der Ansatz selbst ist als „best-effort“-Ansatz zu verstehen, d.h., es kann aufgrund der Charakteristika von MANETs in der Regel nicht garantiert werden, dass eine solche Steuerungsnachricht auch beim Empfänger ankommt, sondern lediglich durch eine gute Parameterwahl die Wahrscheinlichkeit einer erfolgreichen Übertragung erhöht werden. Abbildung 4.4 verdeutlicht die beiden Phasen der Übermittlung grafisch.

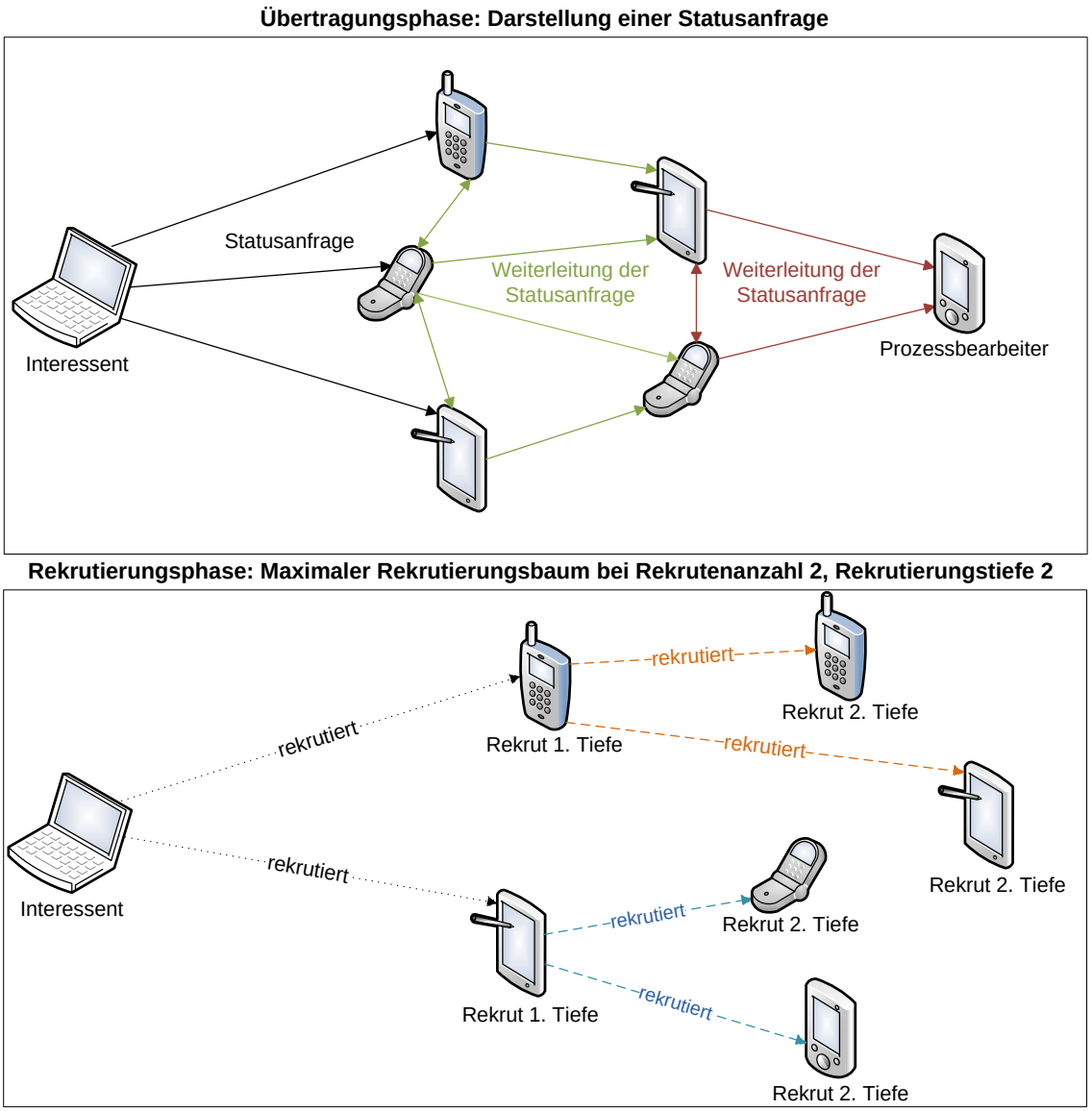

Abbildung 4.4: Darstellung der Phasen der Prozessüberwachung

- **Übertragungsphase**: In dieser Phase sendet der jeweilige Interessent (bzw. Rekrut) die Steuerungsnachricht an alle für ihn erreichbaren Knoten. Jeder Knoten, der diese Anfrage erhält, sendet diese wiederum an alle seine Nachbarn, sodass im Idealfall alle zu diesem Zeitpunkt erreichbaren Knoten die Nachricht erhalten. Bekommt der Sender nach einer bestimmten festgelegten *Wartezeit* keine Antwort vom Prozessbearbeiter, so sendet er die Steuerungsnachricht erneut und wartet wiederum das durch den Parameter *Wartezeit* festgelegte Zeitintervall. Die Anzahl der Sendeversuche - also wie oft der Interessent (bzw. Rekrut) nach einer spezifischen Wartezeit Steuerungsnachrichten versendet - wird wiederum durch den Parameter *Sendeversuche* festgelegt. Sollte nach dem letzten Versuch ebenfalls keine Antwort ankommen, so wird die Rekrutierungsphase durchgeführt.

- **Rekrutierungsphase**: Die Rekrutierungsphase wird durch die vom Interessenten spezifizierten Parameter *Rekrutenanzahl* und *Rekrutierungstiefe* charakterisiert. Die *Rekrutenanzahl* beschreibt, wie viele Endgeräte rekrutiert werden, die ihrerseits den ursprünglichen Interessenten unterstützen, indem sie die Übertragungsphase durchlaufen. Hat die *Rekrutenanzahl* beispielsweise den Wert 5, so sucht sich der Interessent 5 Endgeräte, die im Namen des ursprünglichen Interessenten periodisch Steuerungsnachrichten übermitteln. Die *Rekrutierungstiefe* wiederum beschreibt, ob die Rekruten nach erfolglosem Abschluss der Übertragungsphase wiederum Rekruten akquirieren. Ist die *Rekrutenanzahl* 5 und die *Rekrutierungstiefe* 2, so bedeutet dies folglich, dass maximal *1+5+5*5=31* Endgeräte die Übertragungsphase durchlaufen. Die 1 steht in dieser Rechnung für den ursprünglichen Interessenten, die 5 wiederum für seine 5 Rekruten, von denen jeder wiederum bis zu 5 Rekruten rekrutiert.

  Sollte auch die Kombination der beiden Phasen zu keiner Antwort des Prozessbearbeiters führen, wird die Übertragung der Steuerungsnachricht eingestellt, wobei der ursprüngliche Interessent prinzipiell die Möglichkeit besitzt, einen neuen Übertragungsversuch mit anderen (oder auch den gleichen) Parametern zu beginnen. Sofern eine der Nachrichten den Prozessbearbeiter erreicht, schickt dieser wiederum eine Antwort nach dem gleichen Schema (d.h. auf Basis der beiden Phasen), wobei er die durch den Interessenten in der Steuerungsnachricht mit übermittelten Parameter verwendet. Damit das Netz in diesem Fall nicht unnötig mit weiteren Nachrichten überlastet wird, stoppt jeder Rekrut des Interessenten beim Eintreffen einer Antwort des Prozessbearbeiters seine Arbeit. Zur Bestätigung der erhaltenen Antwort des Prozessbearbeiters sendet der Interessent lediglich eine weitere Bestätigung, auf die wiederum die Rekruten des Prozessbearbeiters ihre Arbeit einstellen, also das Versenden der Antwort sowie die Rekrutierung. Erhält der Interessent diese Antwort nicht, so endet die Übermittlung der Antwortnachricht, nachdem die Rekruten der letzten Ebene des Prozessbearbeiters die Übertragungsphase durchlaufen haben.

Ziel der Rekrutierungsphase ist es, sofern die Übermittlungsphase keinen Erfolg birgt, über die Rekrutierung, die letztendlich eine Art von Migration ist, die Erfolgsaussichten zu erhöhen. Da Nachrichten energieintensiv sind, sollten gerade die Parameter *Rekrutenanzahl* und *Rekrutierungstiefe* vom Interessenten mit Bedacht gewählt werden. Im Folgenden werden die einzelnen Nachrichten bzw. der Nachrichtenaufbau verdeutlicht.

4.6.2 Aufbau der Nachrichten

Im Rahmen der Prozessüberwachung lassen sich 4 unterschiedliche Nachrichtentypen unterscheiden:

Steuerungsnachricht: Die Steuerungsnachricht ist diejenige Nachricht, die vom Interessenten und seinen Rekruten im Rahmen der Übertragungsphase versendet wird. Sie beinhaltet den Parameter *Art der Nachricht*, der beschreibt, welche Art von Steuerungsnachricht gemeint ist. Denkbare Werte sind hier z. B. „Abbruchanordnung“ oder „Statusanfrage“. Der optionale Parameter *Attribute* ermöglicht es des Weiteren, weitere Informationen, die die Steuerungsnachricht charakterisieren, zu übertragen. Bei einer „Statusanfrage“ könnte man hierüber beispielsweise den gewünschten Detaillierungsgrad der Statusantwort festlegen. Damit Steuerungsnachrichten unterschieden werden können, beinhaltet die Steuerungsnachricht weiterhin den eindeutigen Identifikator (*SID*). Neben diesen Parametern ist der Identifikator des Interessenten (*AbsenderID*) notwendig, damit der Prozessbearbeiter weiß, zu welchem Endgerät die Antwort geschickt werden soll. Weiterführend ist trivialerweise ebenfalls der eindeutige Identifikator des Prozesses notwendig (*ProzessID*). Der letzte Part der Nachricht ist die so genannte *Prozessbearbeiteranordnung*, die beschreibt, auf welche Art und Weise die Antwortnachricht übermittelt wird. Sie beinhaltet die Parameter *Sendeversuche*, *Wartezeit*, *Rekrutenanzahl* und *Rekrutierungstiefe*. Die ersten beiden Parameter gehören zur Übertragungsphase und beschreiben, wie oft jeder Rekrut maximal die Steuerungsnachricht überträgt und wie lange zwischen zwei Übertragungsversuchen gewartet wird. Die beiden anderen Parameter beziehen sich auf die Rekrutierungsphase und beschreiben, wie viele Rekruten bis zu welcher Tiefe rekrutiert werden.

Steuerungsnachricht:
<Art der Nachricht, Attribute, SID, AbsenderID, ProzessID, <Prozessbearbeiteranordnung>>
Prozessbearbeiteranordnung:
<Sendeversuche, Wartezeit, Rekrutenanzahl, Rekrutierungstiefe>

Listing 4.1: Aufbau der Steuerungsnachricht

Rekrutierungsnachricht des Interessenten: Die Rekrutierungsnachricht beinhaltet neben einigen für die Rekrutierung notwendigen Parametern sämtliche Parameter der bereits beschriebenen *Steuerungsnachricht*. Im Gegensatz zur Steuerungsnachricht beginnt eine Rekrutierungsnachricht mit dem Wort „Rekrutierungsanfrage“ sowie dem Parameter *RekrutenID*, damit der Empfänger der Nachricht weiß, dass er gemeint ist und dass es sich um eine Rekrutierungsanfrage handelt. Zusätzlich beinhaltet die Nachricht einen Abschnitt *Rekrutenanordnung*, die beschreibt, bis zu welcher Tiefe neue Rekruten maximal rekrutiert werden sollen. Damit die Rekrutierung nur bis zur vorgeschriebenen Ebene durchgeführt wird, wird die Rekrutierungstiefe nach jedem Rekrutierungsdurchgang dekrementiert, bis sie bei 0 angekommen ist, was gleichbedeutend damit ist, dass der/-die jeweilige(n) Rekrut(en) dieser Ebene selbst keine weiteren Rekruten rekrutiert bzw. rekrutieren.

Rekrutierungsnachricht des Interessenten:
<"Rekrutierungsanfrage", RekrutenID, <Rekrutenanordnung>, <Steuerungsnachricht>>
Rekrutenanordnung:
<Sendeversuche, Wartezeit, Rekrutenanzahl, Rekrutierungstiefe>

Listing 4.2: Aufbau der Rekrutierungsnachricht des Interessenten

Antwortnachricht: Die Antwortnachricht des Prozessbearbeiters, die in der Übermittlungsphase verwendet wird, beginnt mit dem Parameter *Art der Nachricht*, der je nach Steuerungsnachricht z. B. mit dem Wort „Abbruchbestätigung" oder „Statusangabe" festgelegt wird und die Antwortnachricht charakterisiert. Zusätzlich beinhaltet die Antwort den eindeutigen Identifikator der Steuerungsnachricht (*SID*) sowie den Identifikator des Prozessbearbeiters (*ProzessbearbeiterID*), den Identifikator des Prozesses (*ProzessID*) und den Identikator des Empfängers (*EmpfängerID*), also des Interessenten, der die ursprüngliche Steuerungsnachricht verfasst hat. Sofern weitere Informationen mitübertragen werden sollen wie z. B. bei einer Statusanfrage der derzeitige Status der Prozessbearbeitung, beinhaltet die Nachricht einen weiteren Parameter namens *Antwort*. Der zuletzt genannte Parameter kann im Rahmen einer Statusanfrage beispielsweise eine Prozentangabe beinhalten (z. B. 45% fertig), oder sogar die aktuelle Logdatei des Prozesses beinhalten.

Antwortnachricht:
<Art der Nachricht, SID, ProzessbearbeiterID, ProzessID, EmpfängerID, Antwort>

Listing 4.3: Aufbau der Antwortnachricht

Rekrutierungsnachricht des Prozessbearbeiters: Die Rekrutierungsnachricht des Prozessbearbeiters beinhaltet sämtliche Parameter und Informationen, die auch in der zugehörigen Antwortnachricht vorkommen sowie einige weitere. Neben den bereits genannten Parametern beinhaltet die Rekrutierungsnachricht zusätzlich das Wort „Rekrutierungsanfrage" und den Identifikator des Rekruten (*RekrutenID*). Darüber hinaus beinhaltet die Nachricht den Abschnitt *Rekrutenanordnung*, der bereits eingeführt wurde und beschreibt, wie die Antwort übermittelt werden soll.

Rekrutierungsnachricht des Prozessbearbeiters:
<"Rekrutierung", RekrutenID, <Rekrutenanordnung>, <Antwortnachricht>>
Rekrutenanordnung:
<Sendeversuche, Wartezeit, Rekrutenanzahl, Rekrutierungstiefe>

Listing 4.4: Aufbau der Rekrutierungsnachricht des Prozessbearbeiters

4.6.3 Systemspezifische Optimierungen

Der hier beschriebene Algorithmus ist beabsichtigt so allgemein gehalten, dass er prinzipiell in jedem beliebigen MANET eingesetzt werden könnte, unabhängig davon, ob und welche Kontextinformationen aufgezeichnet und verarbeitet werden können. In spezifischen MANETs sind daher durchaus Optimierungen möglich, um die Suche nach dem Prozessbearbeiter zu verbessern und gleichzeitig Nachrichten zu sparen. Werden beispielsweise ohnehin schon periodisch Kontextinformationen ausgetauscht, um z. B. ein Overlay-Netzwerk aufzubauen oder zu erhalten, wie dies im Rahmen der *DEMAC-Middleware* (vgl. 2.3.3) der Fall ist, so können diese Informationen direkt mitverwendet werden. Sofern der Prozessbearbeiter aufgrund der Kontextinformationen bereits bekannt ist, müssen in der Übertragungsphase keine zusätzlichen Nachrichten versendet werden, da der Interessent in diesem Fall bereits anhand dieser Informationen erkennen kann, ob der Prozessbearbeiter erreichbar ist oder nicht. Er könnte sich demnach sinnvollerweise die Sendeversuche sparen und z. B. direkt Rekruten akquirieren, die ebenfalls anhand der bereits vorhandenen Kontextinformationen die Suche ohne zusätzliche Nachrichten durchführen können.

4.7 Aufbau der managementspezifischen Dateien

Nachdem der allgemeine Ablauf im Managementsystem dargelegt wurde, wird nun der Aufbau der Logdatei sowie des Steuerungsdokuments verdeutlicht. Um die relevanten Parameter der beiden Dateien zu verdeutlichen, wurde zu diesem Zweck jeweils ein (zusammengesetztes) *XML-Schema* (*extensible markup language*) spezifiziert. XML bietet sich an, da es eine technologieunabhängige Meta-Sprache ist, die es ermöglicht, die Struktur von Dateien und Dokumenten festzulegen, ohne zum Beispiel Einschränkungen im Hinblick auf die Programmiersprache zu machen. Bezüglich des allgemeinen Konzeptes ist es aber keineswegs notwendig, die Struktur per XML festzulegen oder den hier spezifizierten Aufbau zu verwenden. Die Nutzung der hier spezifizierten Strukturen ist zudem nicht nur im Rahmen der Kooperation über mobile Prozesse denkbar, sondern kann ebenfalls in anderen Kontexten eingesetzt werden.

4.7.1 Aufbau des Steuerungsdokuments

Im Steuerungsdokument wird beschrieben, in welchem Umfang Management durchgeführt werden soll, d.h., es wird mithilfe von spezifischen Parametern festgelegt, welche Informationen geloggt werden sollen und in welchem Maße die Prozessbearbeitung abgesichert und gesteuert wird. Das Steuerungsdokument setzt sich strukturell aus vier Bereichen zusammen. Es beinhaltet jeweils einen Abschnitt für die einzelnen Kernfunktionen des Managements (Logging, Monitoring und Recovery), in dem jeweils die zugehörigen Parameter zur Steuerung des jeweiligen Aspektes spezifiziert werden sowie

einen Abschnitt „Allgemeine Parameter“, in dem zusätzliche Steuerungsparameter und -informationen untergebracht werden, die den anderen drei Teilgebieten übergeordnet sind. Das Steuerungsdokument wurde generisch und erweiterbar gehalten, sodass Änderungen und Erweiterungen möglichst einfach durchführbar sind. Abbildung 4.5 verdeutlicht den allgemeinen strukturellen Aufbau des Steuerungsdokuments, während Anhang A.1.1 den detaillierten Aufbau in Form des entwickelten XML-Schemas verdeutlicht. Im Folgenden werden die vier Teilbereiche im Detail beschrieben:

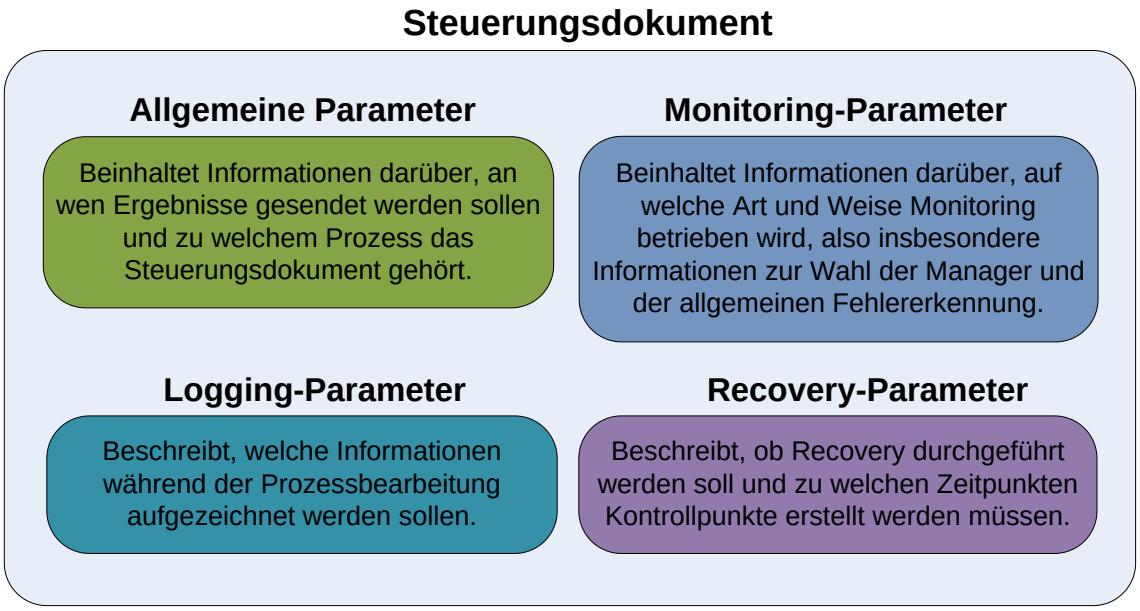

Abbildung 4.5: Allgemeine Struktur des Steuerungsdokuments

Allgemeine Parameter

Um eine Zuordnung des Steuerungsdokuments zu einem Prozess zu ermöglichen, beinhaltet das Dokument den eindeutigen Identifikator des Prozesses (*ProzessID*). Damit ein Prozessbearbeiter weiß, zu welchem/welchen Endgerät/Endgeräten Ergebnisse gesendet werden sollen, kann eine *Empfängerliste* spezifiziert werden. Neben den Identifikatoren der jeweiligen Empfänger (*EmpfängerID*) kann in der Liste festgehalten werden, zu welchen Zeitpunkten Ergebnisse übertragen (*Ergebnisübermittlung -> Zeitpunkte*) und auf welche Art und Weise die Ergebnisse übermittelt werden. Letzteres wird über die Parameter *Sendeversuche*, *Wartezeit*, *Rekrutenanzahl* und *Rekrutierungstiefe* festgelegt, die bereits im Abschnitt 4.6 im Rahmen der Steuerung und Überwachung von Prozessen betrachtet wurden und in diesem Kontext die gleiche Bedeutung haben. Des Weiteren hat der Prozessinitiator die Möglichkeit, über zwei Identifikatorenlisten festzulegen, welche Benutzer bzw. Endgeräte autorisiert sind, den Abbruch der Bearbeitung des Prozesses anzuordnen bzw. den derzeitigen Bearbeitungsstatus zu erfragen (*Abbruchliste*, *Statusliste*). Die Struktur ist auch hier generisch gehalten, sodass die Struktur neben der Abbruch-

und der Statusliste auch um weitere Autorisationslisten erweiterbar ist. Abbildung 4.6 beschreibt die Struktur grafisch.

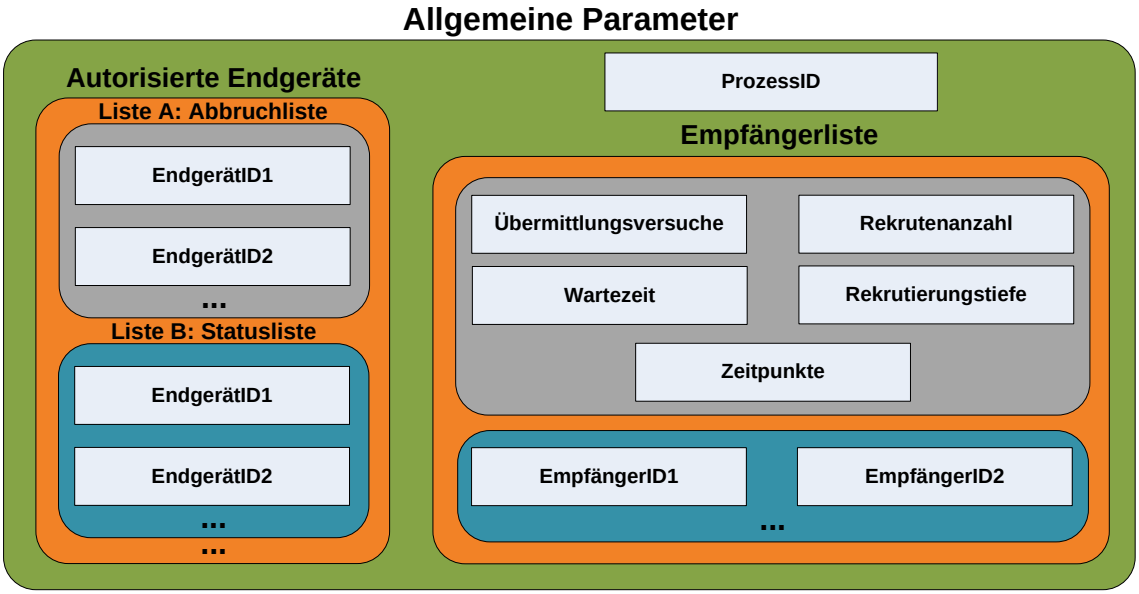

Abbildung 4.6: Allgemeine Parameter des Steuerungsdokuments

Recoveryspezifische Parameter

Im Rahmen der Recovery sind bezüglich des Managements zwei Parameter vom Prozessinitiator festzulegen. Zum einen ist es notwendig zu wissen, ob überhaupt eine Absicherung in Form von Kontrollpunkten stattfinden soll. Diese Wahlmöglichkeit wird über den Parameter *Kontrollpunkte* ausgedrückt. Der zweite Parameter beschreibt wann, also zu welchen Zeitpunkten, Kontrollpunkte erstellt und synchronisiert werden sollen. Da hier viele unterschiedliche Ansätze denkbar sind, wurde dieser Parameter in der XML-Datei ebenfalls generisch bzw. erweiterbar gehalten. Denkbar ist beispielsweise, einen Kontrollpunkt nach jeder n-ten Aktivität zu erstellen oder über eine Liste detailliert zu spezifizieren, nach welcher Aktivität jeweils ein Kontrollpunkt erstellt wird. Die beiden zuletzt genannten Möglichkeiten wurden als Beispiel für den Parameter *Kontrollpunktsynchronisation* umgesetzt (vgl. Anhang A.1.2). Damit im Fall eines Prozessausfalls die Logdaten zum Kontrollpunkt passen, müssen des Weiteren bei jeder Kontrollpunktsynchronisation ebenfalls die aktuellen Logdateien mit den Managern synchronisiert werden. Dies verhindert z. B., dass für bereits durchgeführte Aktivitäten die zugehörigen Logeinträge im Rahmen eines Ausfalls verloren gehen. Abbildung 4.7 beschreibt die Struktur grafisch.

Monitoringspezifische Parameter

Im Hinblick auf das Monitoring muss festgelegt werden, wie viele Manager ein Prozessbearbeiter eines Prozesses zu einem Zeitpunkt haben soll. Dies wird über den Pa-

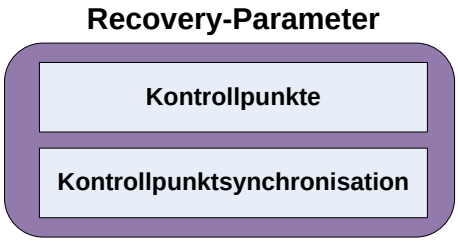

Abbildung 4.7: Recovery-Parameter des Steuerungsdokuments

rameter *Manageranzahl* festgelegt. Hierbei müssen die Implikationen, die eine große bzw. kleine Manageranzahl haben, vom Prozessinitiator berücksichtigt werden (vgl. 4.5). Der Parameter *MinManageranzahl* gibt hierbei an, wie viele Manager sich in Bezug auf den Ausfall des Prozessbearbeiters mindestens einig sein müssen, damit der Prozess neu gestartet wird. Mithilfe dieses Parameters lässt sich die Wahrscheinlichkeit des Auftretens einer Prozessduplikation verringern (vgl. 4.5), wobei ein zu hoher Wert für den Parameter dazu führen kann, dass ein Prozess nicht neu gestartet wird, obwohl der Prozessbearbeiter tatsächlich ausgefallen ist. Bei der Wahl der Belegung dieses Parameters muss folglich für den jeweiligen Prozess abgewägt werden, ob die Wahrscheinlichkeit einer Prozessduplikation oder des Prozessausfalles minimiert werden soll. Ein weiterer wichtiger Parameter ist die so genannte *Kommunikationswartezeit*, die beschreibt, wie lange auf eine managementspezifische Anfrage gewartet werden soll. Ist die Wartezeit beispielsweise 400 ms, so wartet ein Knoten (z. B. ein Prozessbearbeiter), der eine Anfrage an einen anderen Knoten schickt, 400 ms auf eine Antwort und führt, sofern er keine Antwort in diesem Zeitfenster erhält, eine protokollspezifische Aktion aus (z. B. sendet erneut eine Anfrage oder schickt eine Anfrage an einen anderen Knoten). Sofern *Manageranzahl* >= 1 gewählt wird, muss der Prozessinitiator die folgenden Parameter festlegen, damit das Management ordnungsgemäß durchgeführt werden kann:

- *Qualitätsparameter:* Mithilfe des Feldes *Qualitätsparameter* wird festgelegt, nach welchen Kriterien die Manager ausgewählt werden sollen. Möglich sind hier z. B. die Batterielebensdauer oder die Rechenleistung.

- *Heartbeatwartezeit:* Über den Parameter *Heartbeatwartezeit* wird die Zeitspanne festgelegt, die ein Sender von Heartbeat-Nachrichten nach Absenden eines Heartbeats wartet, bis er den nächsten versendet. Eine kleine *Heartbeatwartezeit* impliziert eine höhere Anzahl an gesendeten Heartbeat-Nachrichten pro Zeitintervall, was negative Auswirkungen auf den Energieverbrauch der beteiligten Endgeräte hat, da der Versand und Empfang von Nachrichten energieintensiv ist. Der Vorteil einer kurzen *Heartbeatwartezeit* besteht darin, dass Ausfälle in der Regel schneller erkannt werden und damit früher Recovery-Maßnahmen eingeleitet werden können. Ist die *Heartbeatwartezeit* hoch, so werden Ausfälle ggf. später erkannt, aber das allgemeine Nachrichtenaufkommen ist kleiner.

Abbildung 4.8: Monitoring-Parameter des Steuerungsdokuments

- *derzeitigeEndgeräte:* Für die Koordination zwischen den an einem Prozess beteiligten Endgeräten ist es notwendig zu wissen, welche Manager und welcher Prozessbearbeiter zu einem Zeitpunkt für einen Prozess zuständig sind. So muss ein Manager z. B. sämtliche anderen Manager kennen, damit im Fall eines Ausfalls des Prozessbearbeiters nur ein Manager den Prozess wiederherstellt und nicht alle vorhandenen Manager. Diesen Zweck erfüllt der Parameter *derzeitigeEndgeräte*, in dem die Identifikatoren der beteiligten Endgeräte gespeichert werden und der bei jeder Änderung bezüglich der Konstellation der zuständigen Endgeräte aus genannten Gründen aktualisiert werden muss.

Abbildung 4.8 verdeutlicht die beschriebene Struktur grafisch.

Loggingspezifische Parameter

Über die loggingspezifischen Parameter wird in der XML-Datei festgelegt, welche Informationen aufgezeichnet werden sollen. Dieser Punkt hängt sehr stark vom Prozess und vor allem vom Interesse des Prozessinitiators ab, weshalb im Hinblick auf diesen Aspekt die Struktur der XML-Datei offengehalten wird, indem bei Bedarf ein eigenes Schema eingebunden werden kann, das beschreibt, welche Informationen aufgezeichnet werden sollen. Im Abschnitt 3.4.2 wurden gängige Logeinträge beim Logging im Allge-

meinen und beim Workflow-Management im Speziellen betrachtet, die auch im Rahmen des Loggings mobiler Prozesse relevant sind. Als Beispiel zur Umsetzung des Loggings wurden daher die Felder *Dauer*, *Recoveryanzahl*, *Ergebnisse*, *Prozessbearbeiter* und *Manager* festgelegt (vgl. Anhang A.1.3). Über die Parameter *Dauer* und *Recoveryanzahl* kann der Prozessinitiator festlegen, ob für jede Aktivität die Dauer ermittelt und aufgezeichnet werden soll bzw. ob festgehalten werden soll, wie oft der Prozess nach Fehlern wieder aufgesetzt werden musste. Mithilfe der Felder *Ergebnisse*, *Prozessbearbeiter* und *Manager* kann der Prozessinitiator darüber hinaus bestimmen, ob sämtliche an der Prozessbearbeitung beteiligte Endgeräte (Prozessbearbeiter und/oder Manager) in Form ihrer Identifikatoren geloggt werden sollen, um anhand dieser Informationen beispielsweise eine Abrechnung zu realisieren und ob die Ergebnisse jeder ausgeführten Aktivität, z. B. für spätere Analysen, gespeichert werden sollen. Abbildung 4.9 verdeutlicht die zugehörige Struktur grafisch.

Abbildung 4.9: Logging-Parameter des Steuerungsdokuments

Abbildung 4.10 verdeutlicht die Struktur des Steuerungsdokuments anhand eines vereinfachten Beispiels grafisch. Anhang A.1.4 demonstriert zudem das Beispiel in Form eines XML-Dokuments.

4.7.2 Aufbau der Logdatei

In der Logdatei werden die Loginformationen gespeichert, die vom Prozessinitiator im Steuerungsdokument spezifiziert wurden. Aufgrund des Zusammenhangs zwischen dem Steuerungsdokument und der zugehörigen Logdatei erfordern Änderungen im Schema des Steuerungsdokuments in der Regel auch Änderungen im zugehörigen Schema der Logdatei. Fügt man beispielsweise zum Schema des Steuerungsdokuments einen beliebigen Parameter hinzu, der dem Prozessinitiator die Möglichkeit gibt, zusätzliche bzw.

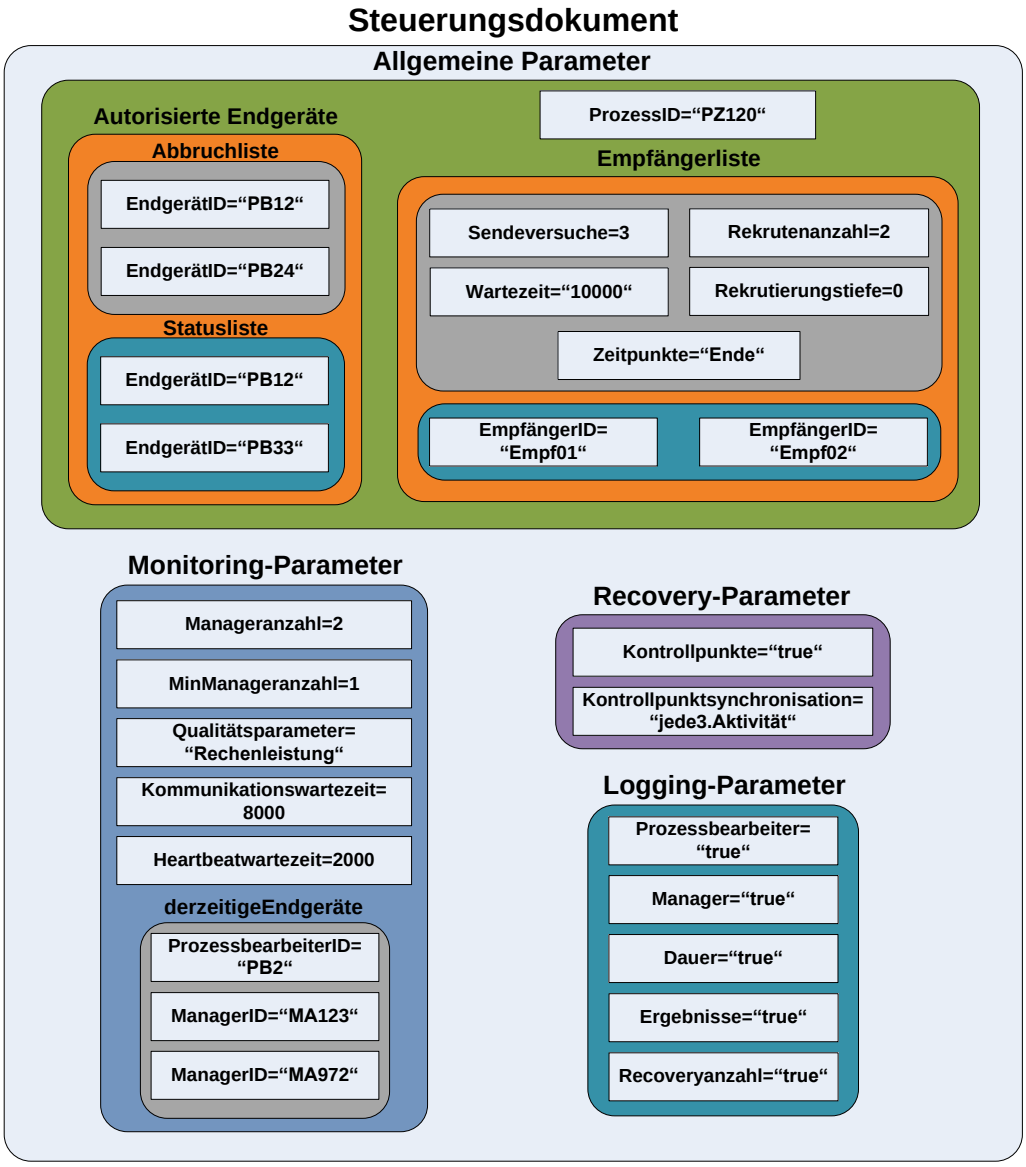

Abbildung 4.10: Beispiel eines Steuerungsdokuments

andere Informationen zu loggen, so muss in der Regel ebenfalls das Schema der Logdatei um diesen Aspekt erweitert werden. Damit derartige Änderungen möglichst einfach durchzuführen sind, beinhaltet das Schema der Logdatei neben einem Abschnitt für allgemeine Parameter lediglich die zwei Teilbereiche *Aktivitäts-Log* (zum Speichern von aktivitätsbezogenen Log-Informationen) und *Prozess-Log* (zum Speichern von prozessglobalen Informationen), die ausgetauscht bzw. leicht erweitert werden können, um beliebige andere Loginformationen festzuhalten. Abbildung 4.11 verdeutlicht den allgemeinen Aufbau der Logdatei.

Für die beiden Teilbereiche *Aktivitäts-Log* und *Prozess-Log* wurden zum bereits vorgestellten XML-Schema des Steuerungsdokuments konforme Beispielbäume entwickelt, die es ermöglichen, die im Steuerungsdokument vom Prozessinitiator festgelegten Infor-

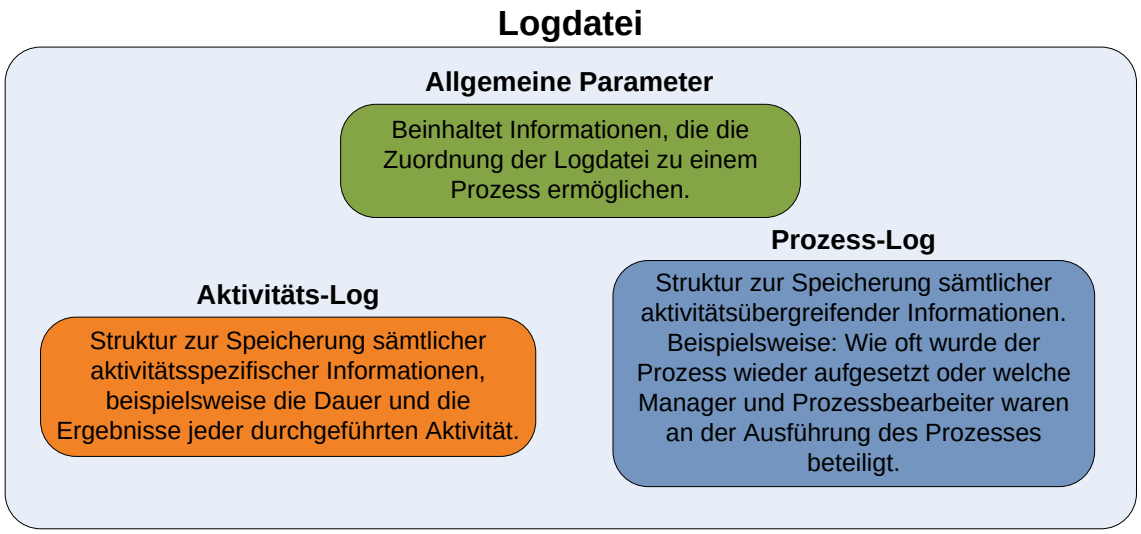

Abbildung 4.11: Allgemeine Struktur der Logdatei

mationen strukturiert zu speichern. Anhang A.2.1 zeigt das XML-Schema der Logdatei, und Anhang A.2.2 ein Beispiel für die Elemente *Aktivitäts-Log* und *Prozess-Log*. Im Folgenden werden die einzelnen Parameter bzw. Felder der resultierenden Logdatei im Detail beschrieben.

Allgemeine Parameter

Damit jede Logdatei ihrem Prozess eindeutig zugeordnet werden kann, beinhaltet das XML-Schema ein Attribut *ProzessID*, in dem der eindeutige Identifikator des Prozesses gespeichert wird.

Aktivitäts-Log

Für jede abgeschlossene Aktivität wird innerhalb des Feldes *Aktivitäts-Log* ein *Aktivität*-Element erzeugt und die im Steuerungsdokument spezifizierten Informationen werden aufgezeichnet. Die Speicherung der durchgeführten Aktivitäten und ihrer Informationen verläuft hierbei chronologisch, d.h., eine Aktivität, die vor einer anderen steht, wurde vor dieser Aktivität durchgeführt.

Eine *Aktivität* enthält die Attribute *AktivitätsID* und *Durchlauf*. Das erste Attribut beschreibt den eindeutigen Identifikator der jeweiligen Aktivität. *Durchlauf* gibt hingegen an, welcher Durchlauf einer Aktivität gemeint ist. Dies ist zur Unterscheidung von Aktivitäten mit gleichem Identifikator notwendig, da Prozesse auch Schleifen von Aktivitäten erlauben und damit Aktivitäten mit dem gleichen Identifikator auch mehrfach ausgeführt werden können und demnach potenziell mehrfach in der Logdatei auftauchen können. Neben den beiden genannten Parametern beinhaltet die Struktur darüber hinaus ein Element *Dauer*, das für jeden Durchlauf einer Aktivität, sofern erwünscht, die

Dauer im Format „Tage:Stunden:Minuten:Sekunden“ speichert sowie eine Menge von *Ergebnis*-Elementen, mit deren Hilfe aktivitätsspezifische Ergebnisse gespeichert werden können. Abbildung 4.12 zeigt die resultierende Struktur eines Aktivitäts-Logs nach beschriebenem Muster.

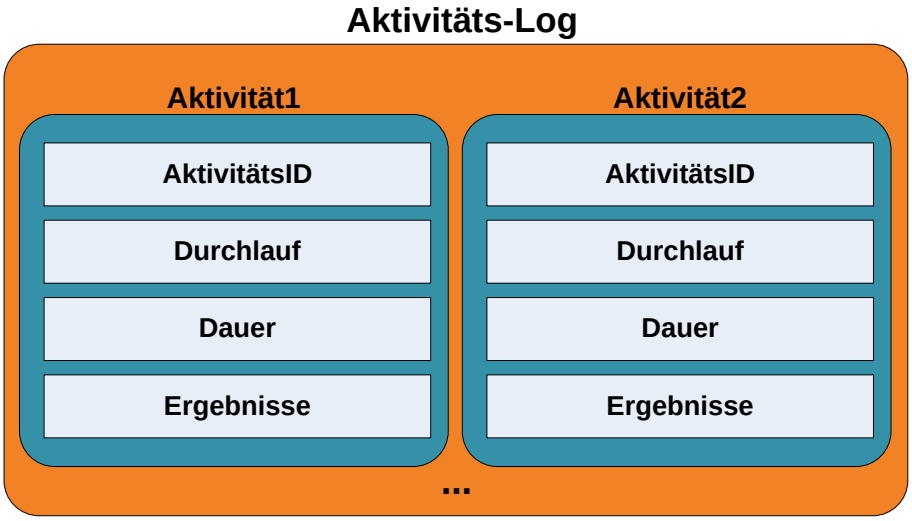

Abbildung 4.12: Aktivitäts-Log der Logdatei

Prozess-Log

Der Beispielbaum für den Abschnitt Prozess-Log beinhaltet einen Parameter mit dem Namen *Prozessbearbeiterliste*, der sämtliche Identifikatoren der Prozessbearbeiter (*ProzessbearbeiterID*), die an der Prozessbearbeitung beteiligt waren, in chronologischer Reihenfolge enthält.

Damit nachträglich ermittelt werden kann, welcher Prozessbearbeiter welche Aktivität durchgeführt hat (z. B. aus rechtlichen Gründen, vgl. 3.4.2), referenziert jedes *Prozessbearbeiter*-Element sämtliche Aktivitäten (*Aktivitätsreferenz*), die vom zugehörigen Prozessbearbeiter durchgeführt wurden. Eine *Aktivitätsreferenz* beinhaltet zu diesem Zwecke jeweils die bereits beschriebenen Parameter *AktivitätsID* und *Durchlauf*. Damit die chronologische Reihenfolge der Prozessbearbeitung nachvollziehbar ist, kann ein Prozessbearbeiter mehrfach in der Liste auftauchen. Führt ein Prozessbearbeiter A beispielsweise 3 Aktivitäten am Stück durch und lässt den Prozess dann zu einem Prozessbearbeiter B migrieren, der eine 4. Aktivität durchführt, so wird der Arbeitsverlauf anschaulich wie folgt gespeichert: A-1,2,3 -> B-4. Sofern der Prozess nun wieder zu Prozessbearbeiter A migriert und dieser eine weitere Aktivität 5 durchführt, so sieht die Speicherung vereinfacht so aus: A-1,2,3 -> B-4 -> A-5.

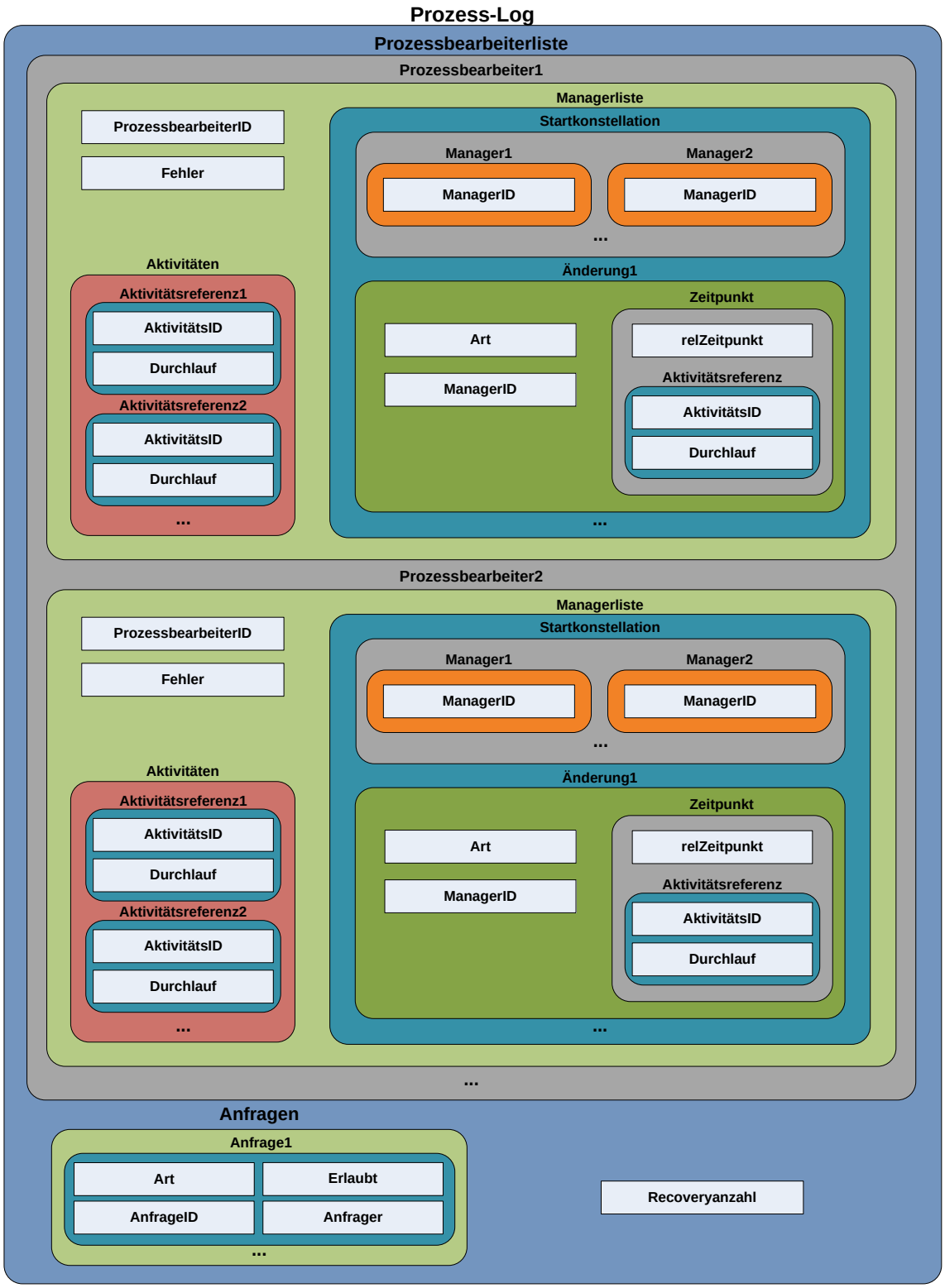

Abbildung 4.13: Prozess-Log der Logdatei

Der boolesche Parameter *Fehler* gibt des Weiteren an, ob der Prozess während der Arbeit des Prozessbearbeiters wieder aufgesetzt werden musste. Ist der Wert „true“, bedeutet dies, dass der Prozessbearbeiter ausgefallen ist und von seinen Managern durch einen

anderen Prozessbearbeiter ersetzt wurde. Ist der Wert hingegen „false“, so bedeutet dies, dass der Prozessbearbeiter den Prozess nach Abschluss der von ihm durchführbaren Aktivitäten an einen anderen Prozessbearbeiter per Migration überreicht hat (oder komplett beendet hat) und damit kein Wiederaufsetzen notwendig war. Auf diese Art und Weise lassen sich folglich „schlechte“ Prozessbearbeiter identifizieren.

Damit nachvollziehbar ist, welche Manager den jeweiligen Prozessbearbeiter in welcher chronologischen Reihenfolge unterstützt haben, wird für jeden Prozessbearbeiter die *Startkonstellation* der Manager und jede *Änderung* gespeichert. Die Startkonstellation eines Prozessbearbeiters sind diejenigen Manager, die er bei „Amtsübernahme“, also beim Start der Prozessbearbeitung, vom vorherigen Prozessbearbeiter (sofern vorhanden) übernommen wurden. Für jede Änderung der Managerkonstellation wird ein *Änderung*-Element erzeugt, das wiederum die Parameter *Art*, *ManagerID* und *Zeitpunkt* beinhaltet. *Art* legt fest, um welche Art von Änderung es sich handelt, mögliche Werte hierfür sind *ManagerEntfernung*, sofern ein Manager entfernt wurde, weil er nicht erreichbar war oder *neuerManager*, falls es sich bei der Änderung um die Wahl eines neuen Managers handelt. Der Parameter *ManagerID* beinhaltet den eindeutigen Identifikator des Managers, auf den sich die Änderung bezieht. *Zeitpunkt* setzt sich wiederum aus den Elementen *relZeitpunkt* und dem bereits beschriebenen zusammengesetzten Parameter *Aktivitätsreferenz* zusammen. Das Element *relZeitpunkt* gibt an, zu welchem relativen Zeitpunkt die Änderung stattgefunden hat. Erlaubte Werte für diesen Parameter sind *start*, *laufend* und *beendet*. *start* bedeutet, dass die Änderung vor Beginn der Bearbeitung der ersten Aktivität durch den Prozessbearbeiter durchgeführt wurde. *laufend* meint, dass die Änderung während der Bearbeitung einer Aktivität durchgeführt wurde und *beendet* bedeutet, dass die Änderung nach der Bearbeitung einer Aktivität durchgeführt wurde. Die beiden zuletzt genannten Werte benötigen zusätzlich eine *Aktivitätsreferenz*, damit festgestellt werden kann, auf welche Aktivität sich der jeweilige Wert und damit die Änderung bezieht.

Für den Spezialfall, dass im Steuerungsdokument festgelegt wurde, dass keine Prozessbearbeiter, sondern nur Manager gespeichert werden sollen, eignet sich die zuvor beschriebene Struktur nicht zur Speicherung. Um diesen Fall abzudecken, wird im Steuerungsdokument auf der obersten Ebene zusätzlich das optionale zusammengesetzte Teilelement *Managerliste* aufgeführt. Es wird analog zum bereits beschriebenen Element *Managerliste* verwendet, wobei auf den inhärenten Parameter *Startkonstellation* verzichtet wird, da dieser Parameter nur sinnvoll ist, wenn ebenfalls die Prozessbearbeiter gespeichert werden.

Neben den genannten Parametern beinhaltet eine Logdatei den zusammengesetzten Parameter *Anfragen*, der Informationen darüber gibt, welche Anfragen (z. B. Abbruch- oder Statusanfrage) für den jeweiligen Prozess eingetroffen sind. Er setzt sich aus den Parametern *Art*, *Erlaubt*, *AnfrageID* und *Anfrager* zusammen. *Art* beschreibt die Art der Anfrage, denkbar sind hierfür z. B. die Werte *Status* für eine Statusanfrage oder *Abbruch* für

eine Abbruchanfrage. *Erlaubt* beinhaltet Informationen darüber, ob die Anfrage durchgeführt wurde. Ein Abbruch wird beispielsweise nur dann zugelassen, wenn der Identifikator des Anfragers sich in der zugehörigen Autorisierungsliste befindet. Der Parameter *AnfrageID* beinhaltet den eindeutigen Identifikator der Anfrage, hierüber kann beispielsweise geprüft werden, ob die jeweilige Anfrage bereits beantwortet wurde und somit verhindert werden, dass gleiche Anfragen, die über unterschiedliche Endgeräte eintreffen, mehrfach beantwortet werden. *AnfragerID* beinhaltet zudem den Identifikator des Anfragers. Zur Feststellung und ggf. Anpassung der Managementparameter für andere Prozesse erlaubt der Parameter *Recoveryanzahl* die Speicherung der Anzahl sämtlicher fehlerspezifischer Wiederherstellungen des Prozesses. Abbildung 4.13 verdeutlicht die beschriebene Struktur des Prozess-Log Abschnitts der Logdatei grafisch.

Mithilfe der hier beschriebenen Struktur und der zugehörigen Parameter kann nach der Prozessbearbeitung folglich festgestellt werden, welcher Prozessbearbeiter welche Aktivitäten durchgeführt hat. Über die Dauer der Aktivitäten lässt sich eine Aussage darüber machen, wie lange der jeweilige Prozessbearbeiter am Prozess mitgeholfen hat, was beispielsweise für Abrechnungszwecke oder spätere Analysen relevant sein kann. Da die Struktur es weiterhin erlaubt festzustellen, welcher Manager in welcher Reihenfolge welche Prozessbearbeiter unterstützt hat, kann über die transitive Relation und auf Basis des *Änderung*-Parameters, die Zeit ermittelt werden, die der Manager für die Sicherung des Prozesses aufgewendet hat. Eine noch „genauere“ Zeitangabe würde aufgrund der potenziell unterschiedlichen lokalen Uhren Zeitsynchronisation erfordern, die in MANET keineswegs trivial ist und außerdem zu einem höheren Nachrichtenaufkommen führen würde. Im Rahmen dieser Arbeit wurde daher auf ein präziseres Verfahren verzichtet, da der Nutzen im Vergleich zum Aufwand in der Regel eher marginal ist. Eine Erweiterung des Konzeptes um diesen Aspekt ist hierbei aber prinzipiell möglich.

Zur besseren Übersicht wurde die grafische Abbildung 4.14 erstellt, die die Struktur der Logdatei anhand eines Beispiels verdeutlicht. Im Beispiel wurden bereits zwei Aktivitäten durch zwei verschiedene Prozessbearbeiter bearbeitet. Der erste Prozessbearbeiter hat zu Beginn seiner Arbeit zwei Manager gewählt, von denen einer während der Übergabe des Prozesses an den zweiten Prozessbearbeiter entfernt und während der Bearbeitung der zweiten Aktivität ersetzt wurde. Bis zu diesem Zeitpunkt ist eine Abbruchanfrage eingetroffen, die abgelehnt wurde, da das Endgerät, das den Abbruch angefordert hat, kein autorisiertes Endgerät war (vgl. 4.10). Anhang A.2.3 zeigt das Beispiel in Form eines XML-Dokuments.

4.8 Umsetzung der Ausfallerkennung

Das Erkennen von Knotenausfällen ist eine wichtige Voraussetzung, um den Ausfall eines Prozessbearbeiters und damit des jeweiligen mobilen Prozesses zu bemerken und auf Basis dieser Informationen geeignete Gegenmaßnahmen wie beispielsweise eine Back-

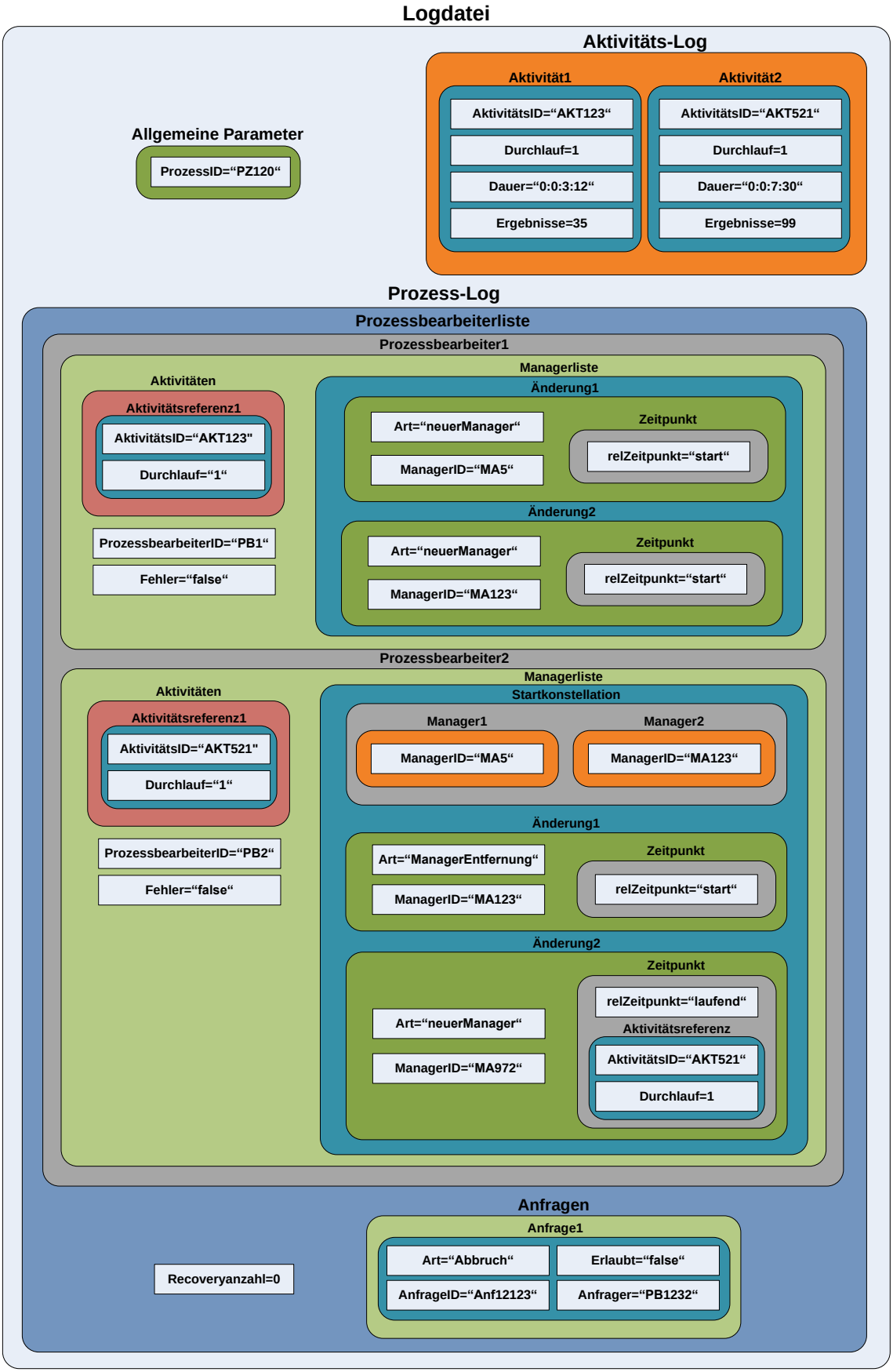

Abbildung 4.14: Beispiel einer Logdatei

ward Recovery durchführen zu können und die Prozessbearbeitung damit trotz Fehler erfolgreich beenden zu können. Wie bereits im Abschnitt 3.4.3 gezeigt wurde, bietet sich zur Ausfallerkennung ein adaptiver Heartbeat-Algorithmus nach dem von *Elhadef et al.* [EAN07] beschriebenem Muster an, da dieser Algorithmus auf MANETs zugeschnitten ist und die Randbedingungen und Gegebenheiten in einem solchen volatilen System berücksichtigt. Der im Folgenden vorgestellte Algorithmus basiert auf dem genannten Algorithmus und wurde an das Szenario der Kooperation über mobile Prozesse angepasst.

4.8.1 Überblick über die Funktionsweise und die spezifischen Anpassungen

Dieser Abschnitt gibt zunächst einen allgemeinen Überblick über die Funktionsweise des originalen Heartbeat-Algorithmus von *Elhadef et al.*, außerdem wird die auf das Management mobiler Prozesse zugeschnittene angepasste Version kurz vorgestellt. Die Veränderungen bzw. Anpassungen des originalen Algorithmus resultieren direkt aus den spezifischen Gegebenheiten und Anforderungen an das Kooperationskonzept der mobilen Prozesse, d.h., die Struktur (u.a. das Rollenkonzept) wird ausgenutzt.

Originaler Algorithmus

Jeder Knoten im Netzwerk sendet periodisch (alle Δ Zeiteinheiten) Heartbeat-Nachrichten an sämtliche direkt, also ohne Zwischenknoten erreichbaren Knoten, wobei die Informationen empfangener Heartbeats in die eigenen zu versendenden Heartbeats integriert werden. Dies wird mithilfe eines Vektors realisiert, der in jede Heartbeat-Nachricht eingebettet wird, wobei dessen Stellen jeweils einen Knoten im Netzwerk repräsentieren und einen Zähler beinhalten, der den höchsten bekannten Heartbeat vom korrespondierenden Knoten beinhaltet. Wird von einem Knoten ein Heartbeat empfangen, so werden die korrespondierenden Einträge im Vektor auf das Maximum des lokalen und des empfangenen Vektors erhöht. Die transitive (*gossip)* Natur dieses Algorithmus führt daher im Optimum dazu, dass jeder Knoten von jedem anderen indirekt oder direkt Statusinformationen erhält. Wird nach einer bestimmten Zeit kein (direkter oder indirekter) Heartbeat empfangen, so verdächtigt der jeweilige Knoten den anderen so lange, bis über einen Heartbeat aktuelle Statusinformationen über den korrespondierenden Knoten eintreffen.

Modifizierter Algorithmus

Im Rahmen des Managements mobiler Prozesse ist es nicht notwendig, dass jeder Knoten Heartbeat-Nachrichten versendet, da in der Regel lediglich der Ausfall eines Knotens mit der Rolle eines Prozessbearbeiters oder eines Managers die Prozessausführung bzw. Prozesssicherung negativ beeinflussen kann. Sämtliche Knoten ohne eine derartige Rolle senden daher keine eigenen Heartbeat-Nachrichten und verdächtigen auch keine anderen Knoten. Damit die Informationen eines Heartbeats dennoch auf transitive Art und Weise im Netz verteilt werden, ist es notwendig, dass jeder Knoten, der eine Heartbeat-

Nachricht empfängt, die nicht ausschließlich für ihn bestimmt ist (ein Prozessbearbeiter sendet z. B. ggf. an mehrere Manager), diese an sämtliche Nachbarknoten weiterleitet. Im angepassten Algorithmus werden folglich Heartbeats weitergeleitet, anstatt die Informationen zu extrahieren und in den eigenen Heartbeat zu integrieren. Sofern ein Prozessbearbeiter oder Manager nach einem bestimmten, über den Algorithmus berechneten Zeitintervall keine aktuelle Heartbeat-Nachricht vom jeweiligen korrespondierenden Knoten erhält, verdächtigt er ihn des Ausfalls und führt je nach Rolle und Manageranzahl die entsprechende protokollspezifische Aktion aus (vgl. 4.4.1) .

4.8.2 Adaptiver Heartbeat-Algorithmus

Die Qualität eines Heartbeat-Algorithmus hängt hauptsächlich davon ab, wie gut die Ankunftszeiten von Heartbeat-Nachrichten geschätzt werden können. [EAN07] Räumt man eine zu hohe Zeitspanne ein, so werden Ausfälle ggf. erst deutlich später erkannt, während eine zu kurze Zeitspanne dazu führen kann, dass Knoten des Ausfalls verdächtigt werden, die eigentlich noch funktionstüchtig sind. Für eine gute Schätzung ist es notwendig, die Entfernung zwischen den Knoten und die Netzwerklast miteinzubeziehen und hierüber die Schätzung dynamisch anzupassen. Um diesen Umstand zu berücksichtigen, wurde im Algorithmus von *Elhadef et al.* in Abhängigkeit davon, ob zwei Knoten, die Heartbeats austauschen, in direkter Nachbarschaft zueinander stehen oder aber mehrere Hops voneinander entfernt sind, unterschiedliche Schätzungsalgorithmen verwendet. Im Folgenden werden zunächst die beiden Teilalgorithmen zur Schätzung vorgestellt und im Anschluss wird die angepasste kombinierte Version verdeutlicht.

Algorithmus für benachbarte Knoten

Für Knoten, die direkt benachbart sind, d.h. ohne Kommunikation über Zwischenknoten kommunizieren können, wurde von *Elhadef et al.* [EAN07] der Algorithmus von *Bertier et al.* [BMS02] verwendet, der zur besseren Schätzung der Ankunftszeiten *Freshness-Points* nutzt. Ein *Freshness-Point* bezeichnet die Schätzung der Ankunftszeit eines Heartbeats unter Berücksichtigung der Ankunftszeiten bereits eingetroffener älterer Heartbeats, was im Allgemeinen zu einer Verbesserung der Genauigkeit der Fehlererkennung führt (vgl. 3.4.3):

Unter der Annahme, das bereits *k* Heartbeats empfangen wurden, wird die Ankunftszeit des *k+1* Heartbeats, der von einem Knoten v gesendet wurde (EA^v_{k+1}), hierbei mithilfe des Durchschnitts der n letzten Ankunftszeiten berechnet. Die folgenden zwei rekursiven Regeln verdeutlichen die Berechnung von EA^v_{k+1}:

> Δ bezeichnet den Zeitraum, den der Heartbeat-Sender nach Versenden eines Heartbeats wartet, bis er die nächste Heartbeat-Nachricht versendet.
>
> A^v_k entspricht der Ankunftszeit des *k-ten* Heartbeats von v.

- **Regel 1:** Sofern weniger als n Heartbeats von v empfangen wurden, wird die Ankunftszeit des nächsten Heartbeats wie folgt berechnet:

$EA^v_{k+1} = U_{k+1} + \frac{k+1}{2} \cdot \Delta$
$U_{k+1} = \frac{A^v_k}{k+1} \cdot \frac{k \cdot U_k}{k+1}$ entspricht der mittleren Ankunftszeit
$U_1 = A^v_0$

Listing 4.5: Berechnung der Ankunftszeit - Gleichung 1

- **Regel 2:** Sofern mehr als n Heartbeat-Nachrichten von v empfangen wurden, schätzt die folgende Gleichung die Ankunftszeit:

$$EA^v_{k+1} = EA^v_k + \frac{1}{n}(A^v_k - A^v_{k-n-1})$$

Listing 4.6: Berechnung der Ankunftszeit - Gleichung 2

Um zu verhindern, dass funktionsfähige Knoten aufgrund von Übertragungsverzögerungen oder Prozessorüberlastungen des Ausfalls verdächtigt werden, wird zusätzlich dynamisch ein Sicherheitsspielraum (auch *safety margin* genannt) ermittelt. Die *safety margin* α^v_{k+1} wird über den *Jacobson Algorithmus* [PA00] wie folgt berechnet:

$\alpha^v_{k+1} = \beta \cdot delay^v_{k+1} + \phi \cdot var^v_{k+1}$
$delay^v_{k+1} = delay^v_k + \gamma \cdot error^v_k$
$var^v_{k+1} = var^v_k + \gamma \cdot (|error^v_k| - var^v_k)$
$error^v_k = A^v_k - EA^v_k - delay^v_k$

Listing 4.7: Berechnung der safety margin

Mithilfe der Jacobson-Schätzung wird die *safety margin* folglich über den Fehler ($error$) in der letzten Schätzung permanent an den Netzwerkstatus angepasst. Parameter γ repräsentiert die Bedeutung der neuen Messung in Beziehung zu den älteren Messungen, $delay$ bezeichnet den Schätzungsspielraum, während var die Abweichung zwischen den Fehlern beschreibt. Die Konstanten β und ϕ erlauben es des Weiteren, die Varianz zu reduzieren. Die Ergebnisse von *Xylomenos* und *Tsilopoulos* [XT06] haben gezeigt, dass für drahtlose Umgebungen die Werte $\beta = 3$ und $\phi = 2$ am besten geeignet sind.

Auf Basis dieser Informationen ergibt sich das nächste *timeout delay* ($\Delta_{to_{k+1}}$), das aktiviert wird, sobald die Heartbeat-Nachricht m^v_k empfangen wird und am nächsten *Freshness-Point* τ^v_{k+1} abläuft:

$$\tau^v_{k+1} = EA^v_{k+1} + \alpha^v_{k+1}$$

Listing 4.8: Berechnung des nächsten Freshness-Points

Dieser Algorithmus wurde von *Elhadef et al.* unverändert für die Berechnung der Heartbeat-Wartezeit von benachbarten Knoten verwendet und wird auch in der an das Management mobiler Prozesse angepassten Algorithmus-Version unverändert übernommen.

Algorithmus für nicht benachbarte Knoten

Für Knoten, die lediglich über Zwischenknoten kommunizieren können, wird zur Schätzung der Ankunftszeit eine Variante des *Friedmann et al.*-Ansatzes [FT05] verwendet. Der Ansatz in der Ursprungsform erlaubt einen Abstand von maximal γ_f Hops zwischen zwei aufeinanderfolgenden Heartbeats mit dem Ziel, durch die Mobilität auftretende längere Routen und Nachrichtenübertragungsfehler zu kompensieren. Das bedeutet: Sofern der Abstand größer als γ_f wird, wird das jeweilige Endgerät des Ausfalles verdächtigt. Das *Timeout-Intervall* ($\Delta_{to_{k+1}}$) wird hierbei jedes Mal, wenn relevante Heartbeats eintreffen, wie folgt gesetzt:

Δ bezeichnet den Zeitraum, den der Heartbeat-Sender nach Versenden eines Heartbeats wartet, bis er die nächste Heartbeat-Nachricht versendet.

δ beschreibt hierbei die erwartete *One-Hop* Netzwerkverzögerung.

$$\Delta_{to}(\gamma_f) = \gamma_f \cdot (\Delta + \delta)$$

Listing 4.9: Berechnung des Timeout-Intervalls

Zu Beginn wird der *timeout* auf $\Delta_{to}(\gamma_0)$ gesetzt. Der Parameter γ_0 bezeichnet die geschätzte maximale Zeit, die vergeht, bis der erste Heartbeat vom entferntesten Endgerät im Netzwerk eintrifft. Eine Berechnung von γ_0 ist in [FT05] zu finden.

Änderungen nach *Elhadef et al.*

Die Verwendung eines festen Abstandes γ_f zwischen zwei aufeinanderfolgenden Heartbeats führt zu schlechten Ergebnissen, da Ausfälle zwischen Knoten, die wenige Hops entfernt sind, ggf. erst spät erkannt werden und Endgeräte, die viele Hops voneinander entfernt sind, einander ggf. verdächtigen, obwohl die Knoten gar nicht ausgefallen sind. [EAN07] *Elhadef et al.* schlagen daher eine dynamische Bestimmung des Abstandes zwischen zwei mobilen Endgeräten γ_f mithilfe einer Funktion *EstimateGap* vor. Für die Funktion können verschiedene Heuristiken verwendet werden: Denkbar wäre beispielsweise ein optimistischer Ansatz, bei dem ein Endgerät, dessen Heartbeat nicht über den kürzesten Pfad ankommt, des Ausfalles verdächtigt wird. Weitere Möglichkeiten sind ein pessimistischer Ansatz, bei dem immer die längste Route berücksichtigt wird oder aber zum Beispiel die Verwendung des durchschnittlichen Abstandes.

Änderungen zur Anpassung an das Management mobiler Prozesse

Neben den genannten Änderungen von *Elhadef et al.* muss im Rahmen der Anpassung des Algorithmus an das Management mobiler Prozesse in MANETs die ursprüngliche Formel leicht verändert werden, da nicht jeder Knoten eigene Heartbeats versendet, sondern Heartbeats weitergeleitet werden:

Δ bezeichnet den Zeitraum, den der Heartbeat-Sender nach Versenden eines Heartbeats wartet, bis er die nächste Heartbeat-Nachricht versendet.

δ entspricht hierbei der erwarteten *One-Hop*-Netzwerkverzögerung.

β entspricht der erwarteten Zeit, die ein Knoten benötigt, um eine Nachricht zu verarbeiten.

$(\Delta + \delta)$ beschreibt die geschätzte maximale Zeit vom Heartbeat-Sender bis zum ersten Zwischenknoten.

$\gamma_f \cdot (\delta + \beta)$ beschreibt die geschätzte maximale Zeit vom ersten Zwischenknoten bis zur Ankunft beim eigentlichen Heartbeat-Empfänger.

$$\Delta_{to}(\gamma_f) = (\Delta + \delta) + \gamma_f \cdot (\delta + \beta)$$

Listing 4.10: Angepasste Berechnung des Timeout-Intervalls

Abbildung 4.15 veranschaulicht die Formel.

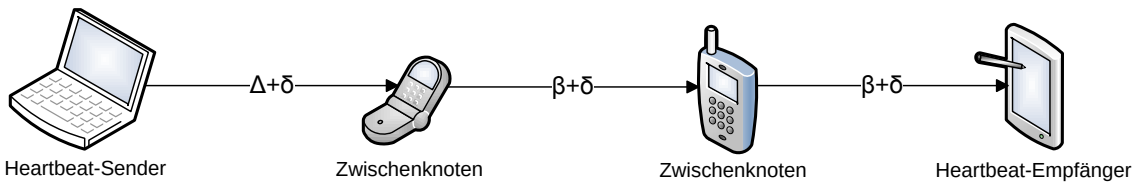

Abbildung 4.15: Beispiel für die Schätzung der Ankunftszeit des nächsten Heartbeats bei nicht benachbarten Knoten

Algorithmus zur Ausfallerkennung beim Management mobiler Prozesse

Wie bereits eingehend beschrieben, kombiniert der Algorithmus von *Elhadef et al.* die beiden zuvor vorgestellten Algorithmen [EAN07]. In diesem Abschnitt wird nicht mehr der originale Algorithmus vorgestellt, sondern direkt die angepasste auf das Management mobiler Prozesse zugeschnittene Variante im Detail verdeutlicht. Der Algorithmus selbst wird hierbei aus der Sicht eines Knotens u beschrieben, der einen Heartbeat von einem Knoten v empfängt. Anhang A.3 beschreibt den Algorithmus in Form eines Pseudocodes, während Abbildung 4.16 den Algorithmus anhand eines Beispiels grafisch veranschaulicht.

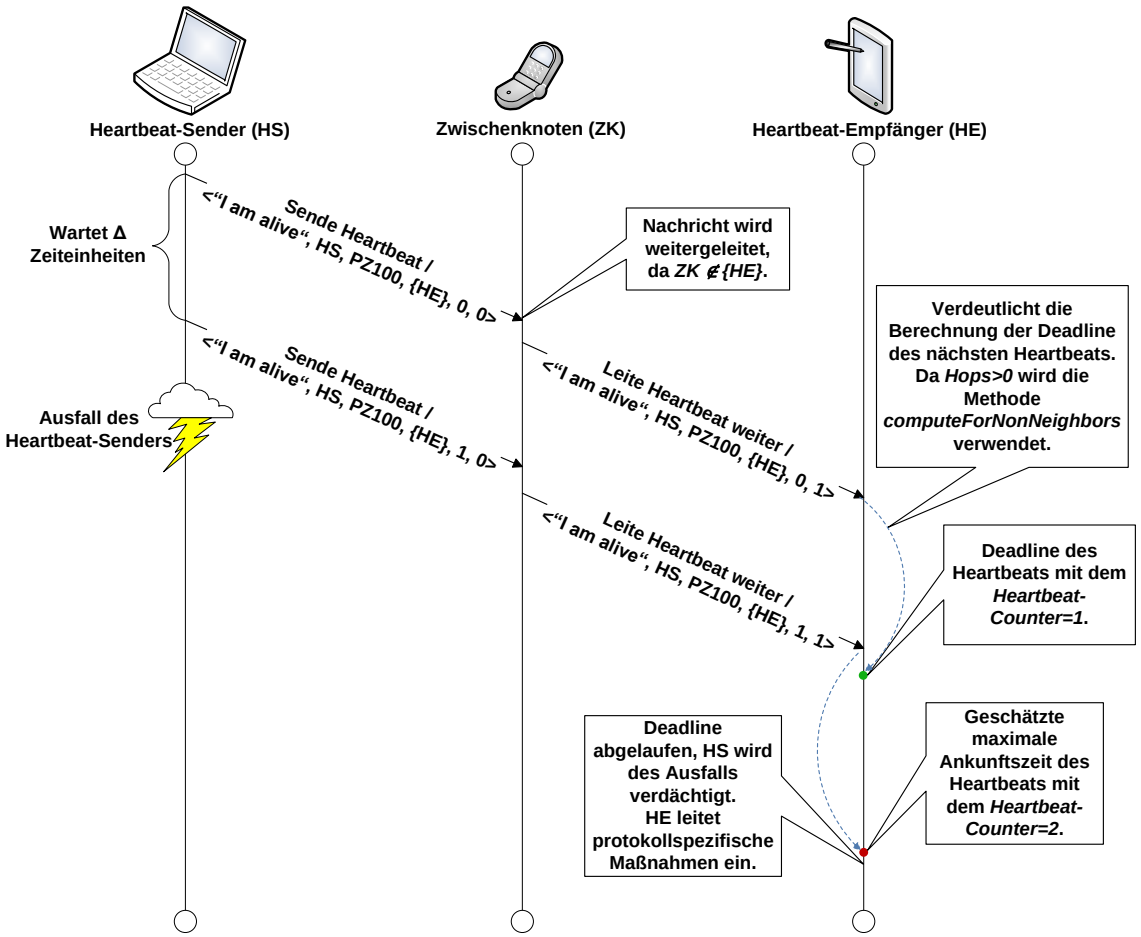

Abbildung 4.16: Algorithmus zur Ausfallerkennung beim Management mobiler Prozesse

- **Aufbau der Heartbeat-Nachrichten**

  Jede Heartbeat-Nachricht enthält einen charakteristischen Text, der es einem Empfänger ermöglicht, die Nachricht als Heartbeat zu identifizieren (*„I am alive“*). Darüber hinaus beinhaltet eine Heartbeat-Nachricht eine Liste von Empfängeridentifikatoren des Heartbeats (*Empfängerliste*) sowie den Identifikator des Senders (*Sender*). Da mehrere Prozesse parallel bzw. nebenläufig im Netz verarbeitet werden können und damit ein Endgerät für einen Prozess als Prozessbearbeiter und für einen anderen Prozess als Manager agieren kann, muss zur eindeutigen Zuordnung des Heartbeats zu einem Prozess für jeden Heartbeat zusätzlich die eindeutige $ProzessID$ übertragen werden. Damit jeder Empfänger alte von neuen Heartbeat-Nachrichten unterscheiden kann, beinhaltet jede Heartbeat-Nachricht eine Zähler-Variable, den so genannten *Heartbeat-Counter*, der bei 0 beginnt und für jede weitere Heartbeat-Nachricht vom Sender inkrementiert wird. Ein Knoten u muss daher für jeden Knoten v, von dem er Heartbeat-Nachrichten erwartet, einen eigenen Zähler $lokalerHeartbeat - Counter_{v,p}$ verwenden (p steht für die *ProzessID*), in dem die höchste Zahl, also die Zahl des jeweils neuesten Heartbeats, gespeichert wird.

Für die Berechnung der Heartbeat-Wartezeit ist darüber hinaus die knotenspezifische Entfernung zwischen Sender und Empfänger notwendig, d.h., jede Nachricht beinhaltet eine Entfernungsvariable *Hops*, die bei 0 beginnt und von jedem Knoten, der eine Nachricht weiterleitet, inkrementiert werden muss. Analog zum *Heartbeat-Counter* muss daher auch eine lokale Variable $lokHops_{v,p}$ verwendet werden, damit auf aus der Mobilität resultierende Veränderungen in der Entfernung adäquat reagiert werden kann.

<"I am alive", Sender, ProzessID, Empfängerliste, Heartbeat-Counter, Hops>

Listing 4.11: Aufbau der Heartbeat-Nachricht

- **Initialisierung:** Nachdem ein Prozessbearbeiter einen Manager gewählt und ihm die aktuellen Statusinformationen und managementspezifischen Dateien übermittelt hat, beginnt die eigentliche Phase des Austausches von Heartbeats zur Ausfallerkennung. Der Knoten *u*, der in diesem Kontext sowohl Manager als auch Prozessbearbeiter sein kann, belegt sämtliche Variablen mit ihren trivialen Werten (vgl. Anhang A.3) und wartet ein im Steuerungsdokument festgelegtes Zeitintervall (Parameter *Kommunikationswartezeit*) auf den ersten Heartbeat:

Sei v ein Knoten der Heartbeats an u sendet und p der zugehörige Prozess.
$lokalerHeartbeat - Counter_{v,p} = 0;$
$lokHops_{v,p} = 0;$
$\alpha_0^{v,p} = var_0^{v,p} = error_0^{v,p} = 0;$
$A_0^{v,p} = var_0^{v,p} = error_0^{v,p} = 0;$
$\tau_0^{v,p} = EA_0^{v,p} = U_0^{v,p} = 0;$

Listing 4.12: Initialisierung des Heartbeat-Algorithmus

- **Sende Heartbeats:** Jeder Prozessbearbeiter sendet periodisch Heartbeat-Nachrichten an seine Manager (sofern vorhanden) und jeder Manager wiederum an seinen Prozessbearbeiter, wobei bei jeder neuen Sendung der jeweilige *Heartbeat-Counter* inkrementiert wird:

```
1 //Heartbeats senden. Für jeden durch Manager geschützten Prozess p
      bezüglich dessen ein Knoten v als Prozessbearbeiter oder
     Manager arbeitet gilt:
2 Task SendHeartbeats(){
3   while(true) {
4     Heartbeat − Counter_p++;
5     Send(<"I am alive", v, p, Empfängerliste, Heartbeat−Counter_p,
          0>);
6     //bis zum nächsten Heartbeat warten
```

```
7       Wait(Δ_p);
8       }
9   }
```

Listing 4.13: Pseudocode des Versendens von Heartbeats

- **Behandlung von Timeouts:** Sofern der Knoten u keine neue Heartbeat-Nachricht von v bekommt, bevor der nächste Freshness-Point $\tau_{k+1}^{v,p}$ abgelaufen ist, verdächtigt der Knoten u den Knoten v des Ausfalls und es werden je nach Rolle und Manageranzahl des Knotens die protokollspezifischen Aktionen zur Behandlung von Knotenausfällen durchgeführt (vgl. 4.4.1).

- **Verarbeitung von eintreffenden Heartbeats:** Bei der Ankunft einer Heartbeat-Nachricht muss ein Knoten u zunächst prüfen, ob die Nachricht für ihn bestimmt ist, d.h., ob sein Identifikator innerhalb der Empfängerliste ist. Hierbei lassen sich zwei Fälle unterscheiden:

  - *Empfänger ist nicht in der Liste:* Ist die Heartbeat-Nachricht nicht für den jeweiligen Empfänger bestimmt, so erhöht er den Parameter *Hops* um 1 und leitet die Nachricht an alle seine Nachbarn weiter. (vgl. Listing 4.14, *Zeile 1-8*)

```
1  //Heartbeats verarbeiten. Verarbeitung eines empfangenen
       Heartbeats vom Sender v, bezüglich eines Prozesses p.
2  Task ProcessHeartbeats(){
3    while(true) {
4      Receive(m_v=<"I␣am␣alive", v, p, Empfängerliste, Heartbeat-
           Counter, Hops>);
5
6      if(u ∉ Empfängerliste) {
7        Send(<"I␣am␣alive", v, p, Empfängerliste, Heartbeat-
             Counter, Hops++ >);
8        }
9      [...]
```

Listing 4.14: Pseudocode des Verarbeitens von Heartbeats für den Fall, dass der Empfänger nicht in der Liste ist

  - *Empfänger ist in der Liste:* Ist die Heartbeat-Nachricht für ihn bestimmt, so erhöht er ebenfalls den Parameter *Hops* um 1 und leitet, falls er nicht der einzige Empfänger in der *Empfängerliste* ist, die Nachricht an alle seine Nachbarn weiter (vgl. Listing 4.15, *Zeile 3-6*). Im Anschluss wird die lokale Ankunftszeit t gespeichert und es wird geprüft, ob die Heartbeat-Nachricht neu oder alt ist (vgl. Listing 4.15, *Zeile 7-30*):

```
1       [...]
2       else {
3         //Nachricht nur weiterleiten, wenn u nicht der einzige
              in der Empfängerliste ist.
4         if(∃x  x ≠ u  x ∈ Empfängerliste ){
5           Send(<"I am alive", v, p, Empfängerliste, Heartbeat−
                Counter, Hops++ >);
6           }
7         t = LocalTime();
8
9         if(Heartbeat − Counter > lokalerHeartbeat − Counter_v){
10          if{Hops == 0}{
11            computeForNeighbors(p, v, Heartbeat − Counter, t);
12            }
13          else{
14            computeForNonNeighbors(p, v, Hops_{v,p});
15              }
16          lokalerHeartbeat − Counter_v = Heartbeat − Counter;
17          StartTimer(τ^{v,p}_{k+1});
18          }
19        //Sofern ein alter Heartbeat über einen längeren Weg
              ankommt, Wartezeit anpassen.
20      else if(Heartbeat − Counter == lokalerHeartbeat − Counter_{v,p}){
21          if(Hops ≠ 0){
22            lokHops_{v,p} = Hops;
23            x_{v,p} = EstHops_{v,p};
24            EstHops_{v,p} = EstimateGap(lokHops_{v,p});
25            if(EstHops_{v,p} > x_{v,p}){
26              τ^{v,p}_{k+1} = τ^{v,p}_{k+1} + (EstHops_{v,p} − x_{v,p}) · (δ + β);
27        }
28            }
29        }
30      }
31    }
32  }
```

Listing 4.15: Pseudocode des Verarbeitens von Heartbeats für den Fall, dass der Empfänger nicht in der Liste ist

Fall A: Falls die Heartbeat-Nachricht neu ist ($lokalerHeartbeat - Counter < Heartbeat - Counter$), wird zunächst geprüft, ob der Heartbeat-Sender ein Nachbar ist (*Hops* = 0 bedeutet Heartbeat-Sender ist Nachbar) oder aber über Zwischenknoten übertragen wurde ($Hops > 0$). Sofern der Heartbeat-Sender ein Nachbar ist, wird der Algorithmus von *Bertier et al.* (vgl. 4.8.2) verwendet, um den nächsten Freshness-Point zu berechnen (vgl. Listing 4.16, *Zeile 1-17*).

Falls die beiden Knoten nicht benachbart sind, wird der für das Management mobiler Prozesse modifizierte Algorithmus von *Friedman et al.* (vgl. 4.8.2) verwendet, bei dem auf Basis der Hopanzahl zwischen Sender und Empfänger dynamisch über die Funktion *EstimateGap* der nächste Timeout berechnet wird (vgl. Listing 4.16, *Zeile 18-22*).

Fall B: Falls die Heartbeat-Nachricht nicht neu ist ($lokalerHeartbeat-Counter =< Heartbeat - Counter$), bedeutet dies, dass ein alter Heartbeat über einen längeren Pfad angekommen ist. Diese Information ist trotzdem nützlich, da impliziert wird, dass, sofern die kürzere Strecke aufgrund von Knoten- oder Übertragungsfehlern ausfällt, ein neuer Heartbeat über diese längere Strecke ankommen kann und daher der nächste Freshness-Point dementsprechend angepasst, also in der Regel verlängert werden sollte (vgl. Listing 4.15, *Zeile 19-30*).

```
1  //Berechnung für benachbarte Knoten
2  Procedure computeForNeighbors(p, v, k_{v,p},t){
3     error_k^{v,p} = t - EA_k^{v,p} - delay_k^{v,p};
4     delay_{k+1}^{v,p} = delay_k^{v,p} + γ · error_k^{v,p};
5     var_{k+1}^{v,p} = var_k^{v,p} + γ · (|error_k^{v,p}| - var_k^{v,p});
6     α_{k+1}^{v,p} = β · delay_{k+1}^{v,p} + φ · var_{k+1}^{v,p};
7
8     if (k_{v,p} < n)  {
9        U_{k+1}^{v,p} = (t+k·U_k)/(k+1);
10       EA_{k+1}^{v,p} = U_{k+1} + ((k+1)/2) · Δ;
11       }
12    else{
13       EA_{k+1}^{v,p} = EA_k^{v,p} + (1/n)(A_k^{v,p} - A_{k-n-1}^{v,p});
14       A_k^{v,p} = t;
15       τ_{k+1}^{v,p} = EA_{k+1}^{v,p} + α_{k+1}^{v,p};
16       }
17 }
18 //Berechnung für nicht benachbarte Knoten
19 Procedure computeForNonNeighbors(p, v,Hops_{v,p}){
20    EstHops_{v,p} = EstimateGap(Hops_{v,p});
21    τ_{k+1}^{v,p} = (Δ_{v,p} + δ) + EstHops_{v,p} · (δ + β);
22 }
```

Listing 4.16: Pseudocode für die Schätzung der Ankunftszeiten von Heartbeats

4.9 Zusammenfassung

In diesem Kapitel wurde das in dieser Arbeit entwickelte Konzept zum Management mobiler Prozesse vorgestellt. Zu Beginn des Kapitels wurden zunächst die getroffenen

konzeptionellen Entscheidungen vorgestellt und begründet. Im Anschluss wurde das Managementsystem in grobkörniger Granularität vorgestellt, um zunächst einen Überblick über den generellen Ablauf und die inhärente Funktionalität zu bekommen. Basierend auf diesem Überblick wurden im Folgenden die Aspekte des Managements im Detail betrachtet.

Neben dem allgemeinen Ablauf der Fehlerbehandlung im Managementsystem wurde ein rekrutenbasierter Ansatz zur Prozessüberwachung vorgestellt, der das Problem der migrationsbezogenen Kontrollabgabe adressiert, also die Steuerung und Überwachung der Prozessbearbeitung, auch nachdem der Prozess den direkten Einflussbereich des jeweiligen Überwachers verlassen hat, erlaubt. Ferner wurde der Aufbau der managementspezifischen Dokumente, namentlich des Steuerungsdokuments und der zugehörigen Logdatei, verdeutlicht. Das Steuerungsdokument ermöglicht es, das Management speziell an die Bedürfnisse des jeweiligen Prozesses bzw. Prozesserzeugers anzupassen, während die Logdatei die strukturierte Speicherung von (Laufzeit-)Informationen ermöglicht, die z. B. für nachträgliche Analysen oder Abrechnungszwecke relevant sein können. Als Kern der Ausfallerkennung wurde zudem ein adaptiver Heartbeat-Algorithmus weiterentwickelt und vorgestellt, der u.a. Aspekte wie die Netzwerklast, die Entfernung zwischen den jeweiligen Knoten sowie bereits erfolgreich empfangene Heartbeats mit in die Berechnung des Timeout-Intervalls einbezieht und dementsprechend eine adäquate und schnelle Ausfallerkennung ermöglicht.

Bei der Entwicklung des Konzeptes wurden die in den vorherigen Kapiteln gewonnenen Erkenntnisse berücksichtigt und wenn möglich bereits bestehende Ideen und anerkannte Umsetzungsmöglichkeiten aus anderen verwandten Bereichen für die Lösung von Teilproblemen in an das Management mobiler Prozesse angepasster Form verwendet. Der entwickelte Managementansatz impliziert weder Annahmen bezüglich der zu Grunde liegenden Netzklasse noch im Hinblick auf die Charakteristika der teilnehmenden Rechner oder der verwendbaren Prozesse, d.h., obwohl das Konzept primär für den Einsatz in mobilen, fehleranfälligen Umgebungen entwickelt wurde, ist es keineswegs hierauf beschränkt, sondern kann dementsprechend auch in beliebigen anderen Kontexten und Umgebungen sinnvoll zur Verbesserung der Kooperationsgüte eingesetzt werden.

Das folgende Kapitel beschäftigt sich mit der prototypischen Implementierung des beschriebenen Konzeptes zur Validierung der zu Grunde liegenden Ideen und Ansätze.

5 Prototypische Implementierung

Nachdem im vorangegangenen Kapitel der entwickelte Ansatz zum Management mobiler Prozesse vorgestellt wurde, beschäftigt sich dieses Kapitel mit der prototypischen Implementierung des Konzeptes in die kontextbasierte DEMAC-Middleware, die im Rahmen des gleichnamigen Forschungsprojektes (*Distributed Environment for Mobility-Aware Computing*) an der Universität Hamburg zum Zwecke der Umsetzung des Kooperationskonzeptes der mobilen Prozesse entwickelt und im Kapitel 2 bereits kurz eingeführt wurde (vgl. 2.3.3).

5.1 Ziele der Implementierung

Einige der wichtigsten Ziele für verteilte Systeme sind allgemein Transparenz (im Rahmen des Managementsystems vor allem Fehlertransparenz), Offenheit, Portabilität, Skalierbarkeit und Erweiterbarkeit (vgl. 2.1.3). Im Hinblick auf die Implementierung war vor allem die aufwandsarme Erweiterbarkeit hoch priorisiert, damit die Managementkomponente möglichst einfach in eine beliebige Umgebung eingebettet und an die dortigen Anforderungen und Wünsche der bzw. des Prozessbearbeiters angepasst werden kann.

Des Weiteren ist die Managementkomponente modular aufgebaut, um eine lose Kopplung der einzelnen Teilkomponenten zu erreichen und dementsprechend einen aufwandsarmen Austausch bzw. eine aufwandsarme Integration neuer Komponenten und Funktionalität zu ermöglichen. Dies ist vor allem wichtig, da je nach Prozess und Anwendungskontext ggf. zusätzliche Informationen in der Logdatei aufgezeichnet und andere Steuerungsmechanismen verwendet werden müssen, die eine Erweiterung des Managementsystems um diese Aspekte erfordern. Zudem wurde bei der Implementierung auf die Verwendung von vollständigen und neutralen Schnittstellen geachtet, um einen hohen Grad an Offenheit zu gewährleisten.

Neben diesen eher allgemeinen Entwicklungszielen war vor allem die Einsetzbarkeit und Performanz in der Klasse der mobilen Systeme mit hoher Priorität versehen, da gerade in diesen Systemen eine Verbesserung der Fehlertoleranz bzw. allgemein der Kooperationsgüte als wichtig herausgearbeitet wurde. Damit dieses Ziel adäquat umgesetzt werden konnte, mussten bei der Entwicklung die identifizierten Restriktionen von mobilen Endgeräten berücksichtigt werden (vgl. 2.2.4). Es wurde vor allem darauf geachtet, dass das Nachrichtenaufkommen im Netz, das zwangsläufig durch die zusätzliche Managementfunktionalität vergrößert wird und damit auch zu einem höheren Energieverbrauch der Endgeräte führt, möglichst gering gehalten wird. Es lassen sich daher sämtliche nachrichtenbezogenen Parameter beliebig konfigurieren, sodass der jeweilige Entwickler einer aufbauenden Anwendung die Möglichkeit besitzt, das Management nahtlos an seine Anwendung sowie die beteiligten Endgeräte anzupassen und weitere Nachrichten (z. B.

durch Erhöhung der *Heartbeatwartezeit*) zu sparen. Abgesehen davon wurde eine Entwicklungsplattform für die Implementierung gewählt, die die Einschränkungen mobiler Endgeräte berücksichtigt.

5.2 Die verwendete Plattform

Der in dieser Arbeit beschriebe Managementansatz wurde in Java, genauer in der Java 2 Micro Edition (*J2ME*) implementiert. Die J2ME ist eine bereits 1999 von Sun Microsystems entwickelte, speziell auf die Bedürfnisse und Restriktionen von mobilen Endgeräten (u.a. geringere Leistungsfähigkeit, hohe Fehleranfälligkeit ; vgl. 2.2.4) zugeschnittene Entwicklungsplattform, die zu diesem Zweck lediglich eine Untermenge der Funktionalität der Java 2 Standard Edition (*J2SE*) bzw. der Java 2 Enterprise Edition (*J2EE*) bereitstellt.

Derzeit existieren für die Micro Edition zwei unterschiedliche *Konfigurationen*, die die Heterogenität zwischen mobilen Endgeräten berücksichtigen und dementsprechend jeweils für Gerätegruppen mit unterschiedlicher Leistungsfähigkeit ausgelegt sind. Die *Connected Limited Device Configuration* (*CDLC*) ist speziell für kleinere Endgeräte (*Low-end consumer devices*) wie z. B. PDAs oder Handys spezifiziert, die über einen Speicher von 128-512KB verfügen und mit 16-32 Bit Prozessoren arbeiten. Die *Connected Device Configuration* (*CDC*), die auch im Rahmen der Implementierung verwendet wurde, ist hingegen eher für leistungsfähigere Endgeräte (*High-end consumer devices*) wie beispielsweise Kopierer, leistungsfähigere PDAs oder SetTop-Boxen ausgelegt, die mindestens 2 MB Speicherplatz sowie einen Prozessor von wenigstens 32-Bit aufweisen.

Zusätzlich existieren diverse *Profile*, die direkt auf der jeweiligen Konfiguration aufsetzen und diese unter anderem um Aspekte wie Datenhaltung, leistungsfähige grafische Oberflächen und spezifische Netzwerkprotokolle erweitern. Im Rahmen der Implementierung der Managementkomponente wurde das auf der Konfiguration des CDC aufsetzende *Personal Profile* verwendet.

5.3 Die Softwarearchitektur des Managementsystems

In diesem Abschnitt wird die Implementierung im Detail beschrieben. Zunächst wird die Kern-Komponente des Managementsystems inklusive der Hauptschnittstelle, die vor allem für Anwendungsentwickler, die eine auf dem Management aufbauende Anwendung entwickeln wollen, essenziell ist sowie die allgemeine Struktur der Management-Komponente beschrieben. Im Anschluss werden jeweils die wichtigsten Pakete und Klassen der einzelnen Managementteilbereiche Monitoring, Logging und Recovery vorgestellt.

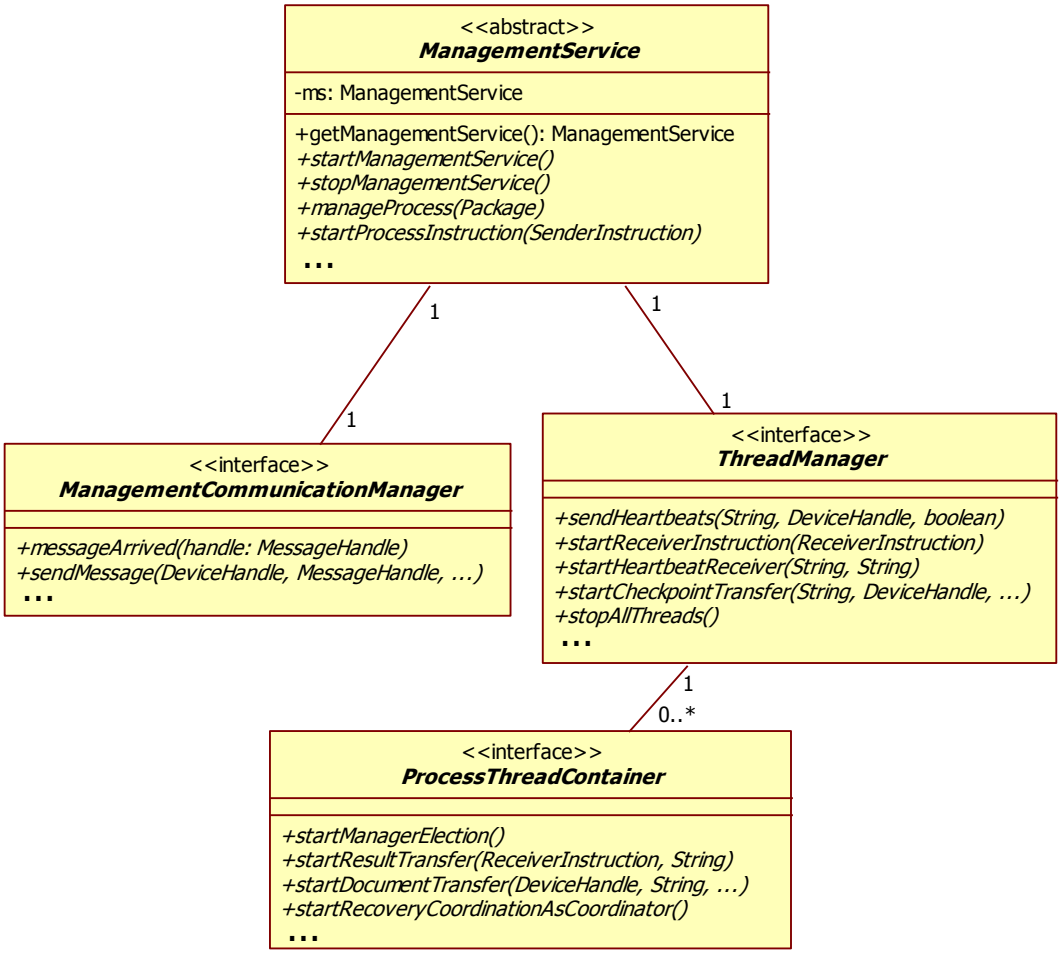

Abbildung 5.1: Struktur der Kernkomponenten des Managementsystems

5.3.1 Der Kern des Managementsystems

Die Klassen und Interfaces der Kernkomponente des Managementsystems befinden sich im Java-Paket *Management.Core*. Der Kern des Managements besteht im Wesentlichen aus der abstrakten Klasse *ManagementService* sowie den Interfaces *ManagementCommunicationManager*, *ThreadManager* und *ProcessThreadContainer*.

Für Anwendungsentwickler, die das Management nutzen wollen, ist die Klasse *ManagementService* am wichtigsten, da sie die zentrale Schnittstelle zum Managementsystem darstellt. Sie ermöglicht den einfachen Zugriff auf die gesamte Managementfunktionalität, ohne dass sich der jeweilige Entwickler weiter mit den Details der Implementierung des Managements beschäftigen muss. Zur Verwendung des Managements müssen beim Start der Prozessbearbeitung lediglich der jeweilige Prozess und das zugehörige Steuerungsdokument übergeben werden. Sofern spezifische Informationen über die Ausführung geloggt werden sollen, müssen diese Informationen (z. B. Ergebnisse von Aktivitäten) ebenfalls von der verwendeten Workflow-Engine an das Managementsystem über-

geben werden.

Die Nachrichtenkomponente wird über den *ManagementCommunicationManager* realisiert, für den genau eine Instanz pro Managementsystem existiert. Seine Aufgabe besteht darin, sämtliche managementspezifischen Nachrichten an andere Endgeräte zu versenden, des Weiteren interpretiert er sämtliche eingehende Managementnachrichten und leitet die Anweisungen an die Ausführungskomponente weiter.

Da beim Ablauf des Managements sehr viele Aufgaben nebenläufig bzw. parallel durchgeführt werden müssen, ist es notwendig, dass die zugehörige Ausführungskomponente - gerade im Hinblick auf einen möglichst geringen Energieverbrauch - die nebenläufigen Aufgaben (im Folgenden Threads genannt), sinnvoll verwaltet und ressourcensparend ausführt. Die genannte Ausführungskomponente wird über den *ThreadManager* und die hiermit verbundenen *ProcessThreadContainer* realisiert. Pro *ManagementService* gibt es genau einen *ThreadManager*. Seine Aufgabe besteht darin, die Anweisungen, die z. B. entweder direkt vom *ManagementService* oder aber über den *ManagementCommunicationManager* eintreffen, dem jeweiligen Prozess zuzuordnen. Dies geschieht, indem er die Anweisung an den betreffenden *ProcessThreadContainer* weiterleitet, der die Anweisung im Anschluss ausführt.

Ein *ProcessThreadContainer* verwaltet sämtliche laufende Threads für einen Prozess. Beim Eintreffen einer Anweisung überprüft er, ob diese schon durch einen laufenden Thread befriedigt wird und erzeugt, sofern notwendig, einen neuen Thread, der die gewünschte Aufgabe durchführt. Auf diese Art und Weise wird verhindert, dass zu viele Threads existieren (pro Prozess existiert z. B. nur eine Managerwahl-Instanz), außerdem lassen sich über die Kombination der *ThreadManager*-Instanz mit den zugehörigen *ProcessThreadContainer*-Instanzen die durchgeführten Aktivitäten ressourcensparend verwalten und dementsprechend sämtliche Threads jederzeit, wenn nötig, gezielt beenden. Abbildung 5.1 gibt einen Überblick über die wesentlichen Klassen des Java-Pakets *Management.Core* und verdeutlicht die Beziehungen zueinander.

5.3.2 Die Monitoring-Komponente des Managementsystems

Die zugehörigen Klassen und Interfaces der Monitoring-Komponente befinden sich im Java-Paket *Management.Monitoring*. Die Monitoring-Komponente umfasst die zentralen Aufgaben Ausfallerkennung, Prozessüberwachung bzw. -steuerung sowie die kontextbasierte Wahl der Manager. Ersteres wird über die Interfaces *HeartbeatReceiver* und *HeartbeatSender* realisiert. Der *HeartbeatSender* ermöglicht das Versenden von Heartbeat-Nachrichten an beliebige Endgeräte unter Berücksichtigung der im zugehörigen Steuerungsdokument festgelegten Parameter (u.a. *Heartbeatwartezeit*). Ein *HeartbeatReceiver* ist für die Verarbeitung eintreffender Heartbeats sowie die Berechnung der maximalen Ankunftszeit des nächsten Heartbeats verantwortlich. Die Komponente ist hierbei so ausgelegt, dass der zu Grunde liegende Heartbeat-Algorithmus leicht ausgetauscht werden kann. Sofern kein Heartbeat innerhalb des berechneten Zeitintervalls eintrifft, wird das kor-

respondierende Endgerät des Ausfalls verdächtigt, der übergeordnete *ProcessThreadContainer* informiert, der wiederum in Abhängigkeit von der Rolle des korrespondierenden Endgeräts die passende Recovery-Maßnahme durchführt.

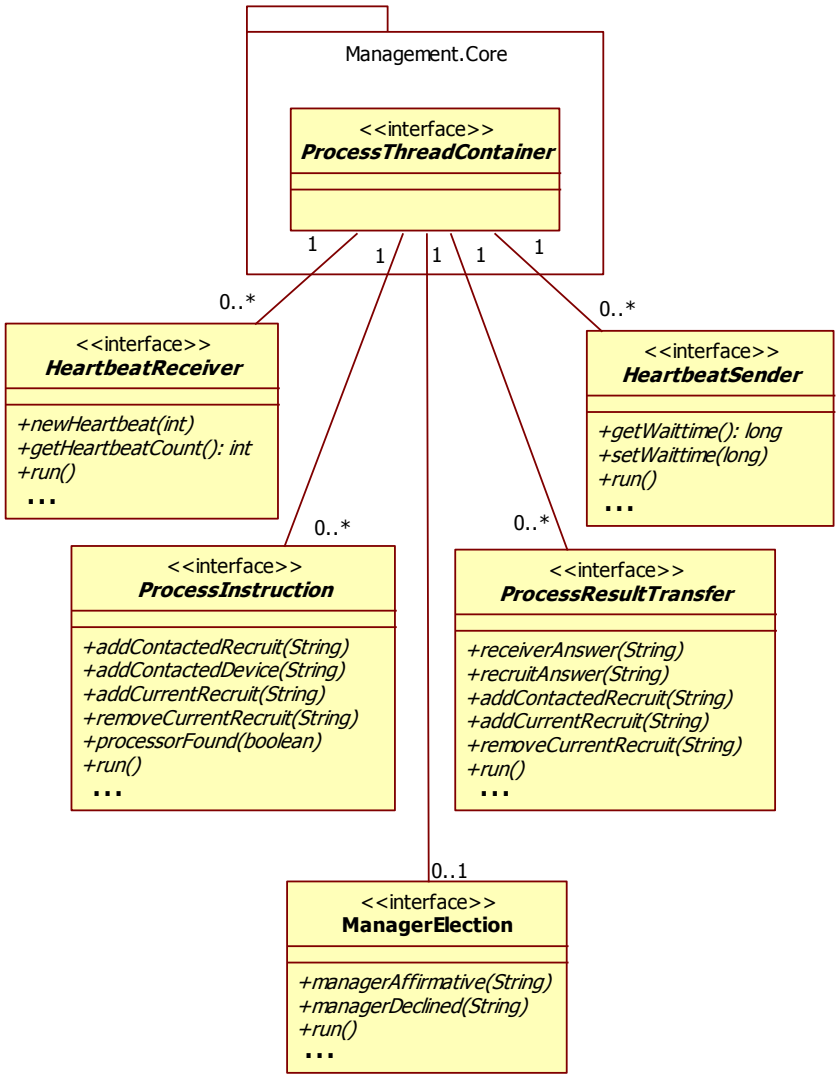

Abbildung 5.2: Struktur der Monitoring-Komponente des Managementsystems

Die Überwachung und Steuerung eines Prozesses wird durch Klassen realisiert, die die Interfaces *ProcessInstruction* und *ProcessResultTransfer* implementieren. Die derzeitige Umsetzung der Überwachungs- und Steuerungskomponente ermöglicht es, die Prozessbearbeitung abzubrechen, den Status der Bearbeitung in Form der aktuellen Logdatei anzufordern oder aber den Identifikator des derzeitigen Prozessbearbeiters zu ermitteln. Die Komponente ist so realisiert, dass eine einfache Erweiterung um weitere beliebige Anfragen möglich ist. Für die Übermittlung einer Anfrage ist es zunächst notwendig, den derzeitigen Prozessbearbeiter ausfindig zu machen. Zu diesem Zweck wird für jede

Anfrage eine *ProcessInstruction*-Instanz erzeugt, die die Suche nach dem Prozess steuert, wobei die Suche auf Basis des in dieser Arbeit entwickelten Rekrutierungskonzeptes verbessert werden kann. Sobald der Prozessbearbeiter die zugehörige Anfrage erhält, führt er sie aus, sofern der Anfrager die Berechtigung für die spezifische Anfrage besitzt, d.h., sofern der Identifikator des Anfragers im Steuerungsdokument bezüglich der Anfrageart vermerkt wurde (vgl. 4.7.1). Für die Übertragung von Ergebnissen erzeugt der jeweilige Prozessbearbeiter bzw. der zugehörige *ProcessThreadContainer* des Anfragers eine *ProcessResultTransfer*-Instanz, die die Übertragung von Ergebnissen nach dem vom ursprünglichen Anfrager gewünschten Muster, wahlweise mit oder ohne Rekruten, durchführt.

Die Managerwahl wird im Rahmen der Monitoring-Komponente über eine Klasse realisiert, die das Interface *ManagerElection* implementiert. Die Wahl erfolgt hierbei unter Berücksichtigung der im Steuerungsdokument spezifizierten Qualitätsparameter und unter Einbeziehung des Parameters *Kommunikationswartezeit*, der die Zeit spezifiziert, die zwischen zwei Wahldurchgängen gewartet wird.

Abbildung 5.2 verdeutlicht den Aufbau und das Zusammenspiel der Interfaces der Monitoring-Komponente.

5.3.3 Die Logging-Komponente des Managementsystems

Die für die Logging-Komponente wichtigen Interfaces und Klassen befinden sich im Java-Paket *Management.Logging*. Im Wesentlichen handelt es sich hierbei um die abstrakte Klasse *ManagementDocumentHandler* sowie die Interfaces *ReadControlDocument*, *ReadLogfile*, *ReadWriteControlDocument* und *ReadWriteLogfile*.

Im Managementsystem existiert genau eine *ManagementDocumentHandler*-Instanz, die die Funktionalität der Protokollierungskomponente und die der Checkpointkomponente realisiert (vgl. 4.1). Im Rahmen der Protokollierungskomponente ermöglicht der *ManagementDocumentHandler* anhand des Identifikators des jeweiligen Prozesses den objektorientierten Lese- und Schreibzugriff auf das zugehörige Steuerungsdokument und die Logdatei, ohne dass der jeweilige Anwender die zu Grunde liegende XML-Struktur der Dokumente kennen und/oder verstehen muss. Damit die XML-Dokumente nicht bei jedem Zugriff geparst werden müssen, werden diese im *ManagementDocumentHandler* beim ersten Zugriff in objektorientierter Form gespeichert und lediglich - wenn notwendig (z. B. bei Änderungen) - persistiert.

Neben dem Zugriff auf die genannten Dokumente ermöglicht die Implementierung im Rahmen der Checkpointkomponente die automatische (auch manuelle) Erstellung von Kontrollpunkten nach dem im Steuerungsdokument vermerkten Muster. Des Weiteren werden die erstellten Kontrollpunkte mithilfe des *ManagementCommunicationManagers* an die zugehörigen Manager weitergeleitet. Die Interfaces *ReadLogfile* und *ReadControlDocument* beschreiben die Schnittstelle für den objektorientierten Lesezugriff der Logdateien und der Steuerungsdokumente. Die von ihnen erbenden Interfaces *ReadWriteLogfile*

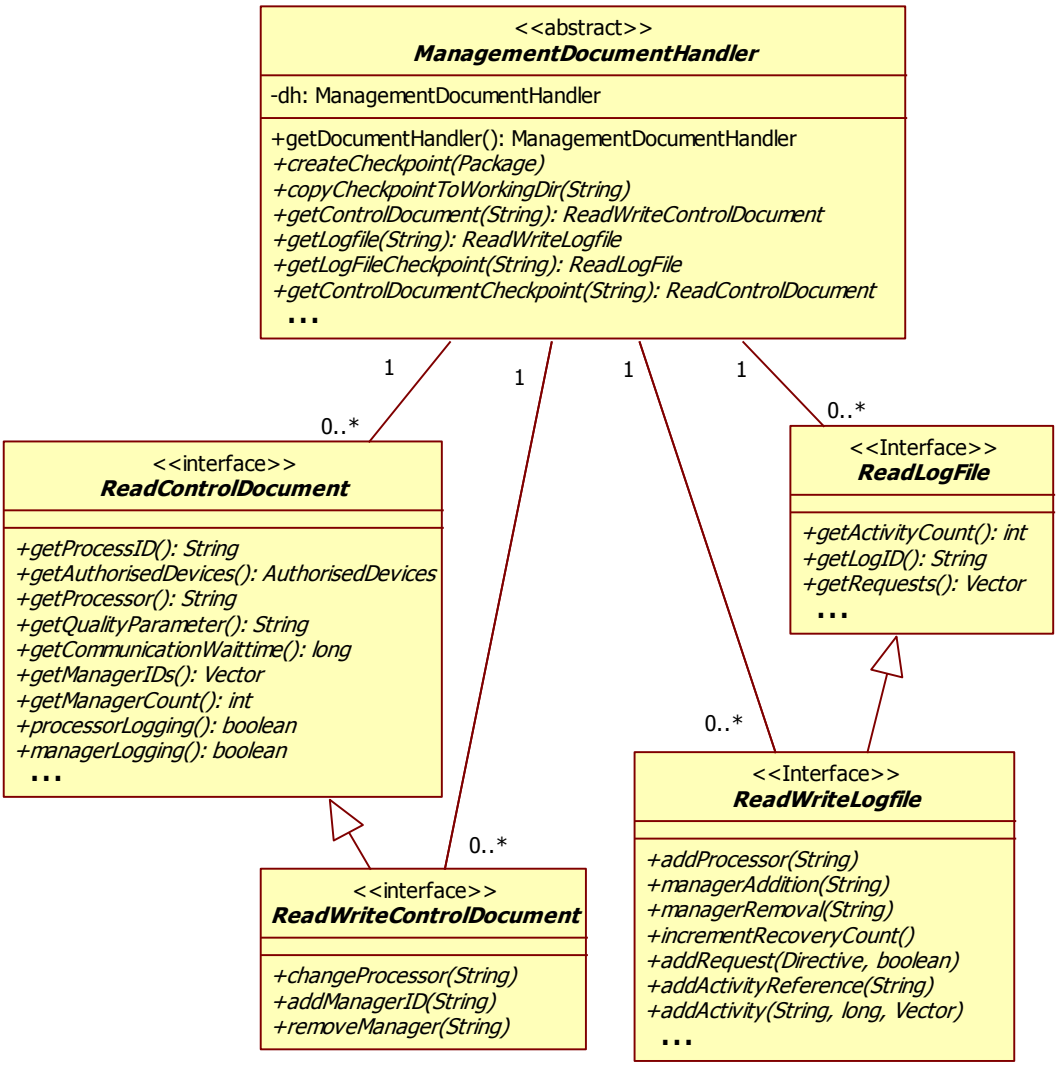

Abbildung 5.3: Struktur der Logging-Komponente des Managementsystems

und *ReadWriteControlDocument* erweitern die Schnittstelle um Schreibzugriff. Die Trennung der Interfaces ist sinnvoll, da z. B. auf Kontrollpunkt-Versionen der Logdatei bzw. des Steuerungsdokuments nicht schreibend zugegriffen werden soll. Da die Interpretation und Veränderung des Prozesses zum Aufgabenbereich der verwendeten Workflow-Engine gehört und dementsprechend nicht Teil des Managements ist, wurde kein objektorientierter Lese- und Schreibzugriff für den Prozess und seine Attribute realisiert. Im Rahmen der Erstellung eines Kontrollpunktes wird daher lediglich eine Kopie des aktuellen Prozesses vom *ManagementDocumentHandler* gespeichert und bei Bedarf, d.h. im Falle einer Recovery an die zugehörige Workflow-Engine übergeben. Abbildung 5.3 verdeutlicht die Struktur der Logging-Komponente.

5.3.4 Die Recovery-Komponente des Managementsystems

Das Java-Paket *Management.Recovery* enthält sämtliche Interfaces und Klassen, die für die Recovery-Komponente notwendig sind. Sie umfasst sämtliche Funktionalität, die notwendig ist, um einen Prozess nach Ausfall des Prozessbearbeiters unter Absprache mit sämtlichen anderen Managern wiederherzustellen.

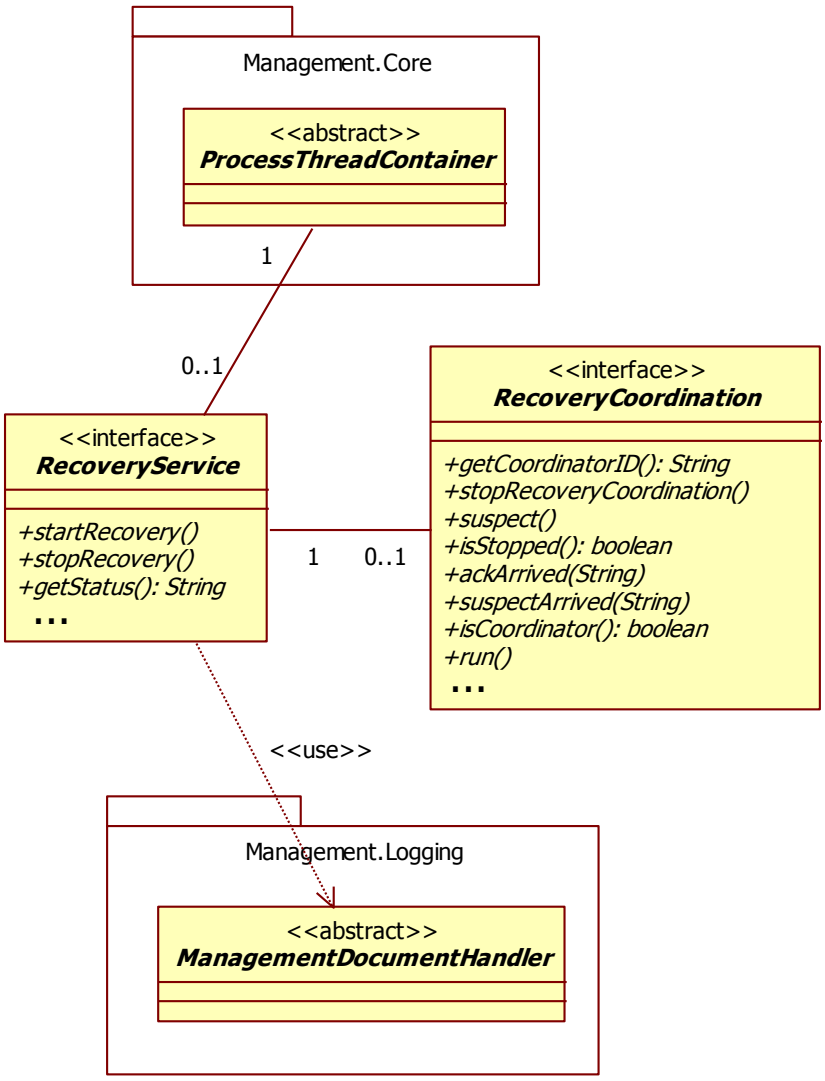

Abbildung 5.4: Struktur der Recovery-Komponente des Managementsystems

Realisiert wird die Komponente im Wesentlichen durch Klassen, die die Interfaces *RecoveryService* und *RecoveryCoordination* implementieren. Eine Recovery wird eingeleitet, sofern die Heartbeat-Komponente des Monitoring-Bereichs keinen Heartbeat innerhalb der berechneten maximalen Ankunftszeit erhält. In dem Fall erzeugt der *ProcessThreadContainer* für den jeweiligen Prozess eine *RecoveryService*-Instanz (sofern für den Prozess noch keine vorliegt) und startet über diese Instanz die Recovery. Der *RecoveryService* prüft

im Anschluss zunächst anhand des zugehörigen Steuerungsdokuments, ob weitere Manager existieren. Sofern derzeit keine weiteren Manager für den Prozess existieren und im Steuerungsdokument vermerkt wurde, dass ein Prozess auch ohne Absprache mit weiteren Managern wiederhergestellt werden darf (Parameter *MinManageranzahl*), wird der Prozess direkt auf Basis des neuesten Kontrollpunktes, den der *RecoveryService* sich vom *ManagementDocumentHandler* besorgt, neu gestartet. Sofern für den Prozess zum jeweiligen Zeitpunkt weitere Manager existieren, erzeugt der *RecoveryService* eine *RecoveryCoordination*-Instanz und führt diese als Recovery-Koordinator aus. Die Koordinations-Komponente befragt im Anschluss sämtliche Manager bezüglich der Ausfallverdächtigung des Prozessbearbeiters nach dem im Abschnitt 4.4.1 beschriebenem Muster. Falls während der Koordination kein neuer Heartbeat eintrifft und sich ausreichend Manager einig sind, wird der Prozess auf Basis des neuesten Kontrollpunktes wiederhergestellt.

5.4 Integration in die DEMAC-Middleware

Die Managementkomponente setzt auf der bereits bestehenden DEMAC-Middleware auf und ist folglich in der Schichtenarchitektur oberhalb der DEMAC-Services anzusiedeln. Hierbei verwendet das Managementsystem einen Teil der von der DEMAC-Midldeware bereitgestellten Services, namentlich den *Context Service*, den *Process Service* und den *Asynchronous Transport Service*, um die Managementaufgaben adäquat zu erfüllen.

Der *Context Service* wird derzeit im Rahmen des Managements verwendet, um Manager anhand von Qualitätsaspekten (z. B. Rechenleistung, Batterielebensdauer) zu wählen und im Rahmen der Überwachung eines Prozesses Nachrichten zu sparen, indem die vom *Context Service* bereitgestellten Informationen über erreichbare Endgeräte mitgenutzt werden. Der *Process Service* übernimmt die Bearbeitung des jeweiligen Prozesses, wobei das Management diese überwacht, d.h. zum einen Informationen über den Bearbeitungsstatus in der Logdatei festhält und zum anderen sicherstellt, dass der Prozess adäquat durch Manager gesichert wird, wobei des Weiteren, sofern notwendig, steuernd in die Bearbeitung eingegriffen wird. Die Nachrichtenkomponente des Managementmoduls baut auf dem *Asynchronous Transport Service* auf, d.h. sie verwendet die bereits bestehende Nachrichtenfunktionalität, außerdem analysiert die Nachrichtenkomponente mithilfe des Listener-Konzeptes zusätzlich den Nachrichtenverkehr, der durch die DEMAC-Middleware erzeugt wird und nutzt diese Informationen, um das Management und die Prozessbearbeitung zu steuern.

Der jeweilige Anwendungsentwickler muss sich daher lediglich mit der bereits beschriebenen Schnittstelle der Managementkomponente vertraut machen, seinen Prozess und das Steuerungsdokument übergeben und kann zusätzlich zur im Steuerungsdokument beschriebenen Managementausprägung jederzeit über die angebotene Monitoring-Funktionalität die Prozessbearbeitung manuell steuern und überwachen. Abbildung 5.5 verdeutlicht die Integration und das Zusammenspiel der einzelnen Schichten bzw. Kom-

ponenten.

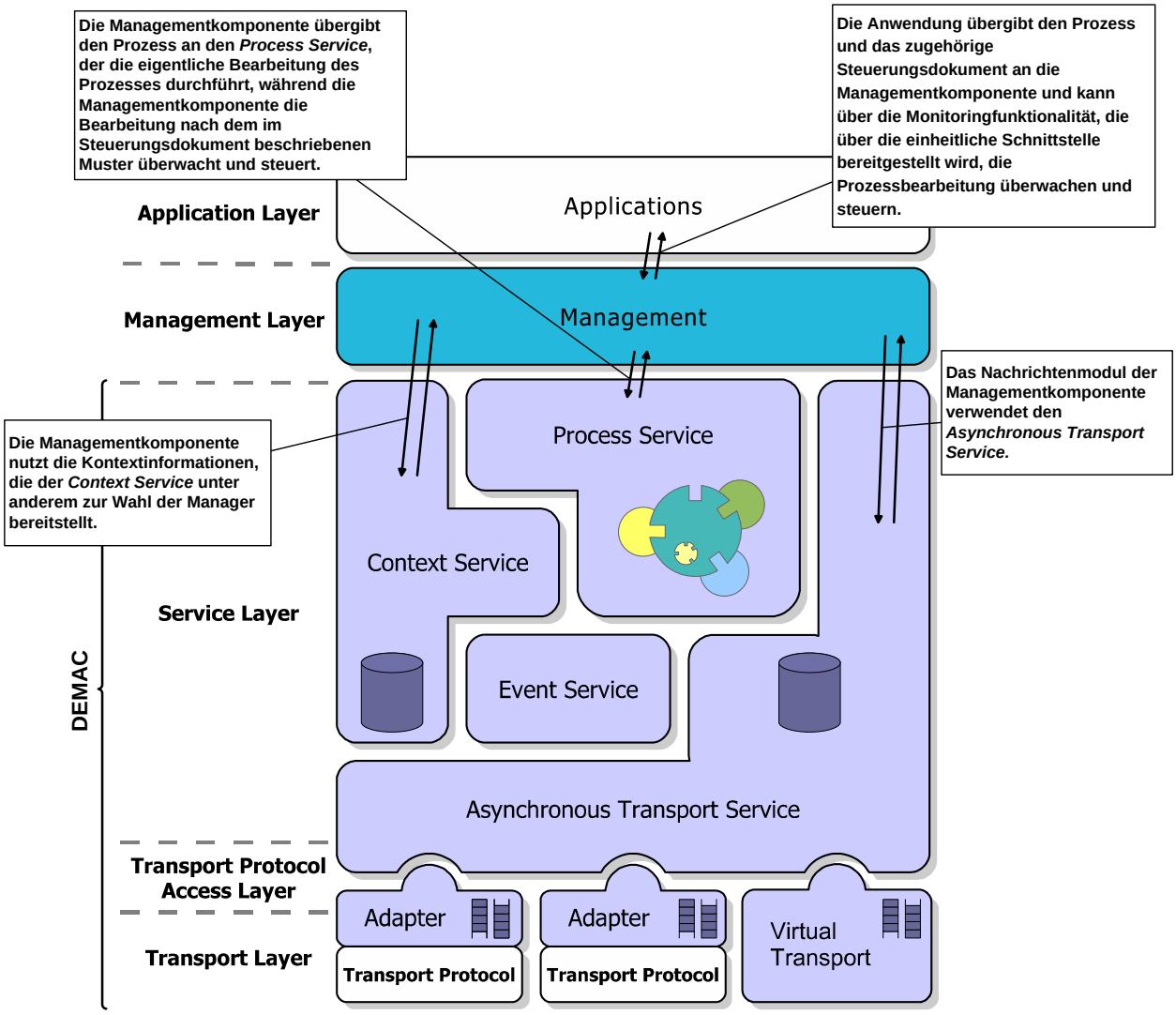

Abbildung 5.5: Integration des Managements in die DEMAC-Middleware

5.5 Test der Implementierung

Zur Überprüfung der Implementierung sowie zur Validierung der inhärenten Konzepte wurden diverse Testprozesse erzeugt. Hierbei wurde darauf geachtet, dass die Prozesse sich dahingehend unterscheiden, dass möglichst das gesamte Spektrum der Interaktionsmöglichkeiten der mobilen Prozesse abgedeckt wird. Zur Überprüfung der Ausfallerkennung sowie der hierauf aufbauenden Recovery-Funktionalität wurden sämtliche Testprozesse mehrfach unter Verwendung unterschiedlicher Managementparameter (u.a. *Manageranzahl* und *Heartbeatwartezeit*) getestet und zufällige Ausfälle der Teilnehmer (Manager und Prozessbearbeiter) simuliert.

Während der Ausfall des Prozessbearbeiters ohne den Einsatz der Managementkomponente zum Verlust der gesamten, bis zu diesem Zeitpunkt getätigten Arbeit führt, konnte im Rahmen der Nutzung der Managementkomponente jeder Prozess unter Berücksichtigung des neuesten Kontrollpunktes genau dann erfolgreich wiederhergestellt werden, wenn noch mindestens ein Manager zum Zeitpunkt des Ausfalles des jeweiligen Prozessbearbeiters zur Verfügung stand. Da der Ausfall von Managern indirekt über die Heartbeat-Komponente nach einem von der gewählten *Heartbeatwartezeit* abhängigen Zeitfenster bemerkt und über die Managerwahl-Komponente kompensiert wird, müssen sämtliche Manager und der Prozessbearbeiter demzufolge nahezu zeitgleich ausfallen, damit ein Fall eintritt, bei dem die Managementkomponente den Prozess nicht adäquat wiederherstellen kann. Die Wahrscheinlichkeit eines solchen Ereignisses lässt sich zudem u.a. über die Parameter *Manageranzahl* und/oder *Heartbeatwartezeit* weiter verringern.

Im Rahmen der Überprüfung der Prozessüberwachungskomponente und dem hierin verankerten Rekrutierungskonzept wurden zudem zu unterschiedlichen Zeitpunkten von unterschiedlichen Endgeräten aus Status-, Such- und Abbruchanfragen gesendet (vgl. 4.6), die, sofern der jeweilige Prozessbearbeiter erreichbar war (d.h. sobald eine Route zwischen Anfrager und Prozessbearbeiter bestand), entsprechend der Spezifikation korrekt beantwortet wurden. Neben den bereits verdeutlichten Aspekten wurde die Logging-Komponente unter Berücksichtigung der genannten Testprozesse sowie unterschiedlicher Steuerungsdokumente überprüft. Hierbei unterstützt die Logging-Komponente die Aufzeichnung sämtlicher im Konzept beschriebener Aspekte sowie die automatische (und manuelle) Erstellung von Kontrollpunkten entsprechend der im Steuerungsdokument vorliegenden Spezifikation.

5.5.1 Beispiel einer Managementanwendung

Im Rahmen der Überprüfung und Validierung der Implementierung wurde eine auf dem Management aufbauende Anwendung entwickelt. Die Anwendung selbst nutzt hierbei die öffentliche Schnittstelle, über die die Managementfunktionalität für Anwendungsentwickler bereitgestellt wird und ist dementsprechend unabhängig von der eigentlichen Implementierung. Abbildung 5.6 zeigt einen Ausschnitt der entwickelten Managementanwendung, die im Folgenden näher erläutert wird.

Der in der Abbildung als *Bereich 1* gekennzeichnete Abschnitt ermöglicht die Übergabe des auszuführenden Prozesses und des zugehörigen Steuerungsdokuments an das Managementsystem sowie die Ausführung der eigentlichen Bearbeitung. Neben dem Label *Process ID* wird zudem der eindeutige Identifikator des Prozesses aufgeführt. *Bereich 2* ermöglicht die manuelle Steuerung und Überwachung des Prozesses mithilfe von spezifischen Anfragen (z. B. Statusanfrage, Abbruchanordnung oder Suchanfrage). Hierzu muss die Art der Anfrage, der Identifikator des Prozesses sowie ein eindeutiger Request-Identifikator, der die Unterscheidung von getätigten Anfragen ermöglicht, angegeben werden. *Bereich 3* ermöglicht zudem die optionale beliebig parametrierbare Unterstüt-

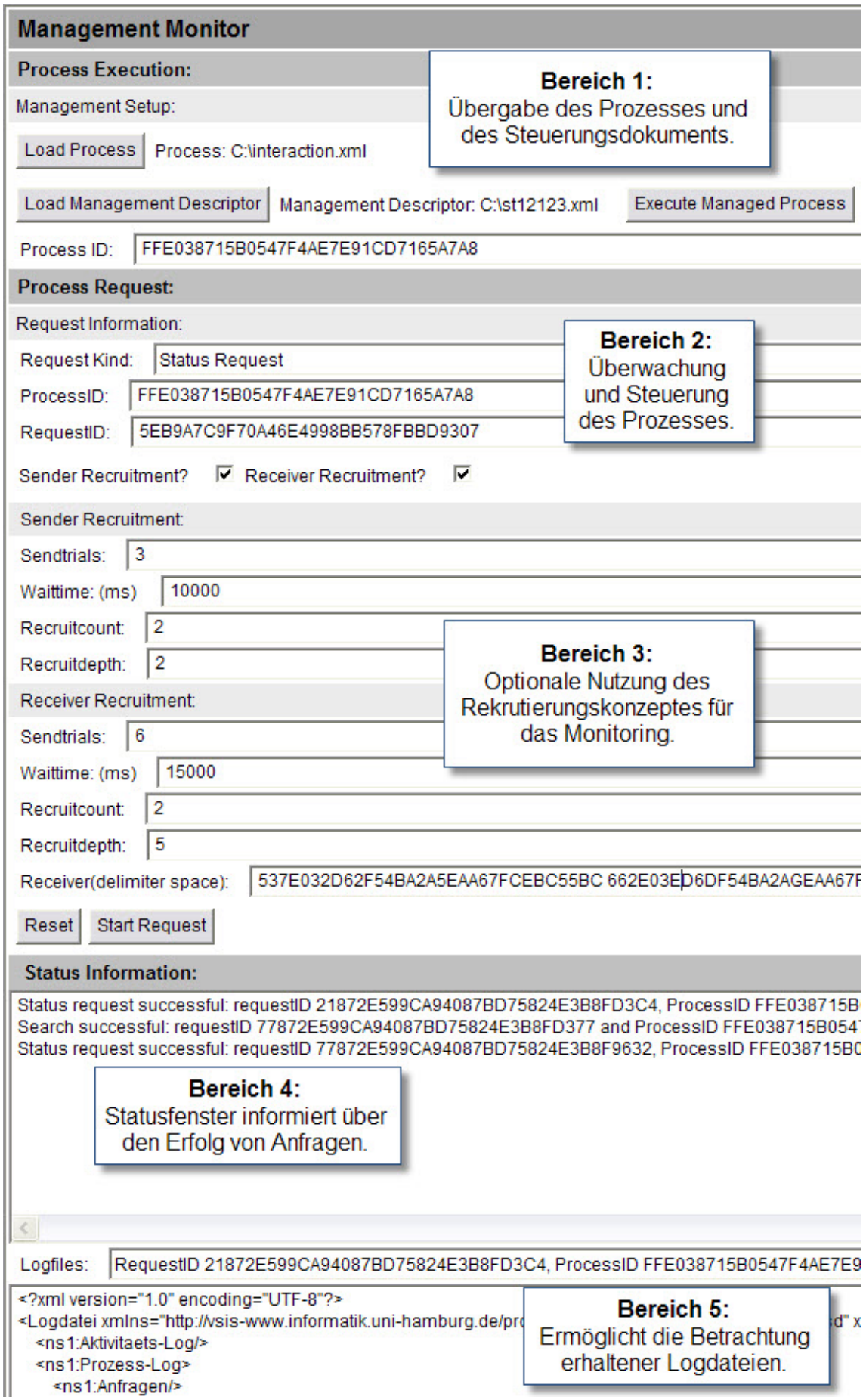

Abbildung 5.6: Ausschnitt einer Managementanwendung

zung der Anfrage- und Antwortübermittlung mithilfe des in dieser Arbeit entwickelten Rekrutierungskonzeptes. *Bereich 4* kennzeichnet das Statusfenster in dem die Ergebnisse der getätigten Anfragen und Anordnungen vermerkt werden, während im *Bereich 5* sämtliche Logdateien, die als Ergebnis einer Statusanfrage erhalten wurden, per Dropdown-Menü ausgewählt und betrachtet werden können.

5.5.2 Betrachtung des Nachrichtenaufkommens

Im vorherigen Abschnitt wurde verdeutlicht, dass die in dieser Arbeit entwickelten Konzepte sinnvolle Maßnahmen umfassen, die Aspekte wie die Nachvollziehbarkeit oder allgemein die Qualität der Kooperation in mobilen Systemen deutlich verbessern. Neben der Umsetzung der eigentlichen Funktionalität war eines der zentralen Ziele bei der Entwicklung die Sicherstellung, dass die Managementkomponente möglichst ressourcensparend ausgeführt und dementsprechend auch von leistungsarmen, mobilen Endgeräten eingesetzt werden kann (vgl. Kapitel 5.1). Einen großen Einfluss auf den Energiehaushalt der beteiligten Endgeräte haben die Nachrichtengröße und -anzahl [CDK05], die im Rahmen des Managements versendet werden müssen, um die gewünschte Funktionalität zu erbringen. Beide Parameter wurden unter anderem kleingehalten, indem die Informationen, die die DEMAC-Middleware bereits akquiriert und bereitstellt, adäquat mitgenutzt werden, um die notwendigen Managementaufgaben durchzuführen. Im Detail wird dies durch die Verwendung des zu Grunde liegenden *TransportService*, *EventService*, *KontextService* und des *ProcessService* der genannten Middleware realisiert (vgl. 2.3.3). Tabelle 5.1 verdeutlicht die Anzahl und Größte der Nachrichten, die für die wichtigsten managementspezifischen Interaktionen benötigt werden.

	gesamte Nachrichtenanzahl	**Nachrichten des Prozessbearbeiter**	**Nachrichten je Manager**	**Nachrichten-größe[1]**
Managerwahl	2*Manageranzahl	Manageranzahl	1	32 Bytes
Heartbeats	2*Manageranzahl pro Heartbeatwartezeit	Manageranzahl pro Heartbeatwartezeit	1 pro Heartbeatwartezeit	36 Bytes
Migration	(2*Manageranzahl)+2	2+Manageranzahl	1	32 Bytes
Kontrollpunkt-synchronisation	2*Manageranzahl	Manageranzahl	1	16 Kilobytes
Recovery-Koordination	4*(Manageranzahl-1)	-	2 bzw. 2*(Manageranzahl-1)	40 Bytes

[1] bezieht sich auf einen Testprozess und kann dementsprechend leicht variieren.

Tabelle 5.1: Anzahl der durch das Management versendeten Nachrichten in Abhängigkeit zur Manageranzahl

Sämtliche durch das Management versendeten Nachrichten sind größentechnisch (abgesehen von der Kontrollpunktsynchronisation) im unteren Byte-Bereich anzusiedeln, also sehr klein und dementsprechend relativ ressourcen- bzw. energiesparend. Die Nachrichten, die bei der Kontrollpunktsynchronisation versendet werden, befinden sich in der Regel im unteren Kilobyte-Bereich, da im Rahmen des Austausches der Kontrollpunkte

das zugehörige Steuerungsdokument, die Logdatei und der Prozess inklusive der aktuellen Statusinformationen übertragen werden müssen. Dementsprechend hängt die Nachrichtengröße hier also direkt von den drei genannten Dokumenten ab, wobei auch diese Nachrichten als relativ klein zu klassifizieren sind. Die eigentliche Nachrichtenanzahl für die jeweilige Interaktionsart ist vom Parameter *Manageranzahl* abhängig. Je mehr Manager den Prozessbearbeiter schützen, desto mehr Nachrichten müssen bezüglich der jeweiligen Interaktion versendet werden. Neben der Manageranzahl beeinflusst die festgelegte *Heartbeatwartezeit* ebenfalls das Nachrichtenaufkommen. Beide Parameter können vom Prozessinitiator beliebig festgelegt werden, wobei u.a. Informationen über die Umgebung, beteiligte Endgeräte und über den Prozess mit einbezogen werden sollten. Hierbei muss berücksichtigt werden, dass eine geringe Manageranzahl und eine hohe *Heartbeatwartezeit* ein geringes Nachrichtenaufkommen (und auch eine geringere Wahrscheinlichkeit der Prozessduplikation) implizieren, gleichzeitig aber Ausfälle später erkannt werden und die Wahrscheinlichkeit, dass der Prozess aufgrund des gleichzeitigen Ausfalles aller Manager nicht wiederhergestellt werden kann, steigt. Der Initiator muss also abwägen, welches der konträren Ziele höhere Priorität für den jeweiligen Prozess besitzt und dementsprechend das Steuerungsdokument spezifizieren.

5.6 Zusammenfassung

Im vorliegenden Kapitel wurde die prototypische Implementierung des in dieser Arbeit entwickelten Managementkonzeptes detailliert beschrieben. Zu Beginn des Kapitels wurden zunächst die allgemeinen Ziele der Implementierung verdeutlicht, die den gesamten Entwicklungsprozess charakterisiert haben. Zentrale Punkte waren hierbei vor allem die ressourcensparende Umsetzung der Managementkomponente sowie die einfache und aufwandsarme Erweiterbarkeit und Portabilität. Im Anschluss wurde eine geeignete zielkonforme Plattform ermittelt, die sich für die Implementierung gut eignet, also vor allem die Umsetzung der Ziele erleichtert. Basierend auf den Erkenntnissen wurde eine passende Softwarearchitektur erarbeitet und die wichtigsten Komponenten sowie die Abhängigkeiten und deren Zusammenspiel verdeutlicht. Die Komponente wurde so entwickelt, dass sie keineswegs auf mobile Prozesse oder eine spezifische Prozessbeschreibungssprache beschränkt ist, sondern durchaus auch aufwandsarm in anderen Kontexten eingesetzt werden kann. Im letzten Abschnitt wurde der Test der Implementierung erläutert. Im Rahmen der Überprüfung der Implementierung zur Validierung des Konzeptes wurde gezeigt, dass die Ideen adäquat umsetzbar und auch zur Verbesserung der Kooperation in restriktiven mobilen Umgebungen geeignet sind. Das folgende Kapitel stellt das Fazit dieser Arbeit dar und gibt einen Ausblick, um zu verdeutlichen, welche Aspekte in aufbauenden Arbeiten sinnvoll behandelt werden können.

6 Fazit und Ausblick

Die stetige technologische Entwicklung ermöglicht es den Menschen im 21. Jahrhundert, ortsungebunden mit einer Vielzahl heterogener elektronischer Geräte auf drahtlose Art und Weise zu kommunizieren und zu interagieren. Neben den Vorteilen, die eine derartige Mobilität ermöglicht, müssen bei der Anwendungsentwicklung die Restriktionen berücksichtigt werden, die bei mobilen Endgeräten im Vergleich zu stationären leistungsfähigeren Endgeräten inhärent sind und entsprechend angepasste Maßnahmen erfordern. Die Verwendung bestehender Konzepte aus dem stationären Bereich ist dementsprechend aufgrund der unterschiedlichen Voraussetzungen in der Regel nicht angebracht, vielmehr sind adäquate auf mobile Systeme und ihre Fehleranfälligkeit zugeschnittene Konzepte und Mechanismen essenziell, die Aspekte wie u.a. die geringere Leistungsfähigkeit berücksichtigen, um trotzdem einen möglichst hohen kooperativen Nutzen in derartigen volatilen Systemen zu erreichen. Für die Abwicklung komplexer Aufgaben bietet sich hierzu als Grundlage das in früheren Arbeiten entwickelte Kooperationskonzept der mobilen Prozesse an. Das Ziel dieser Arbeit bestand darin, aufbauend auf diesem Konzept Methoden und Mechanismen zu entwickeln, die die Kooperationsgüte inklusive wichtiger Aspekte wie Nachvollziehbarkeit und Fehlertoleranz über mobile Prozesse unter Berücksichtigung der genannten Restriktionen verbessern. Im Rahmen der Zielsetzung wurden nach einer Einführung in das Gebiet der verteilten und mobilen Systeme anhand der Restriktionen essenzielle Funktionalitäten (Monitoring, Logging und Recovery) identifiziert und kategorisiert, die innerhalb dieser Arbeit unter dem Begriff Management subsumiert wurden. Bei der Untersuchung bestehender Ansätze für die erarbeiteten Funktionalitäten hat sich gezeigt, dass die meisten bestehenden Ansätze aufgrund der unterschiedlichen Prämissen nicht direkt auf mobile Systeme und die dortigen Gegebenheiten übertragbar sind, sondern lediglich in adaptierter Form eingesetzt werden können.

Als geeignetste Architekturform wurde ein infrastrukturloser Ansatz ermittelt, bei dem die einzelnen Teilnehmer zusätzlich sämtliche Managementfunktionalität bereitstellen und sich selbstständig ohne die Verwendung von ausgezeichneten Server organisieren, da bei einer derartigen Architektur die Vorteile der Mobilität in der Regel nicht beeinträchtigt werden. Basierend auf diesen Erkenntnissen wurde ein Konzept zur Verbesserung der Kooperation in mobilen Systemen entwickelt, wobei wenn möglich bestehende, bereits etablierte Mechanismen in adaptierter, an die Mobilität angepasster Form, verwendet wurden. Das entwickelte Konzept umfasst hierbei Methoden zur automatischen Fehlererkennung und Fehlerbehandlung, zur Überwachung und Steuerung von Prozessen sowie zur Verbesserung der Nachvollziehbarkeit, indem prinzipiell beliebige Informationen über die verteilte Ausführung von Prozessen zur späteren Analyse oder zu Abrechnungszwecken aufgezeichnet, gespeichert und ausgetauscht werden können.

Das entwickelte Konzept ist besonders gut für selbstorganisierende mobile Systeme wie MANETs geeignet, aber zugleich weder in Hinblick auf die Netzart, auf die Prozessart, die Prozessbeschreibungssprache noch auf spezifische Teilnehmer und Endgeräte beschränkt und kann dementsprechend aufgrund der wenigen Prämissen leicht in anderen Umgebungen eingesetzt werden. Im Rahmen der prototypischen Implementierung konnte des Weiteren gezeigt werden, dass das zu Grunde liegende Konzept valide ist und die Kooperationsgüte und -qualität in mobilen Systemen verbessern kann.

Zukünftige Arbeiten könnten den weitläufigen Aspekt der Sicherheit weiter ausführen. Im Rahmen dieser Arbeit wurden im Hinblick auf diesen Aspekt einige rudimentäre Sicherheitsmaßnahmen wie z. B. die Festlegung von Autorisationslisten im Steuerungsdokument integriert, die festlegen, welche Endgeräte bzw. Benutzer die Berechtigung besitzen, Prozessüberwachungs- und Prozesssteuerungsfunktionalität (z. B. Statusabfrage, Abbruchanordnung) für den jeweiligen Prozess zu nutzen. Da das jeweilige Steuerungsdokument im Falle einer Migration mit dem Prozess migriert, besteht die Gefahr, dass das Steuerungsdokument und dementsprechend die Managementparameter vom Prozessbearbeiter unberechtigterweise verändert werden. Zur Vermeidung derartiger Vorfälle könnten beispielsweise Signaturen verwendet werden, die sicherstellen, dass das Steuerungsdokument von unbefugten Zwischenknoten nicht verändert wird bzw. zumindest die Änderung erkannt wird. Da gerade in mobilen Systemen die Kommunikation in der Regel drahtlos abläuft, wird sämtlicher Nachrichtenaustausch per Broadcast realisiert, weshalb zum Schutz vor unbefugtem Zugriff zudem die optionale Verwendung von adäquaten Verschlüsselungsalgorithmen wünschenswert wäre, die die Managementnachrichten und insbesondere die Dokumente schützen. Hierbei sollte allerdings auch berücksichtigt werden, dass Verschlüsselung zusätzlichen Rechenaufwand impliziert, weshalb sichere Algorithmen in mobilen Systemen aufgrund der Beschränkungen mobiler Endgeräte schwerer zu realisieren sind.

Neben den genannten Aspekten sind darüber hinaus auch noch weitere Optimierungen in Bezug auf das Steuerungsdokument denkbar. Derzeit wird das Steuerungsdokument a priori vom Prozessinitiator spezifiziert und ändert sich nicht, abgesehen von den Einträgen über die derzeitigen Endgeräte. Denkbar wäre zum Beispiel, dass das Steuerungsdokument während der Bearbeitung eines Prozesses durch den jeweiligen Prozessbearbeiter angepasst wird, wenn die Umstände dies erfordern. So könnte zum Beispiel nach der Migration eines Prozesses zu einem Endgerät mit geringer Ausfallwahrscheinlichkeit die Manageranzahl und/oder die *Heartbeatwartezeit* reduziert werden, um Energie zu sparen sowie sofern das jeweilige Endgerät eine sehr hohe Ausfallwahrscheinlichkeit besitzt, die genannten Parameter nach oben korrigiert werden, um die Wahrscheinlichkeit eines kompletten Ausfalles zu reduzieren.

Anhang

A.1 Steuerungsdokument

A.1.1 XML-Schema des Steuerungsdokuments

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns="http://vsis-www.informatik.uni-hamburg.de/projects/
       management/steuerungsdokument.xsd" xmlns:xs="http://www.w3.org
       /2001/XMLSchema" xmlns:ns1="http://vsis-www.informatik.uni-hamburg
       .de/projects/management/kontrollpunktsynchronisation.xsd"
       xmlns:ns2="http://vsis-www.informatik.uni-hamburg.de/projects/
       management/logging.xsd" targetNamespace="http://vsis-www.
       informatik.uni-hamburg.de/projects/management/steuerungsdokument.
       xsd" elementFormDefault="qualified" attributeFormDefault="
       unqualified">
3    <xs:import namespace="http://vsis-www.informatik.uni-hamburg.de/
         projects/management/kontrollpunktsynchronisation.xsd"/>
4    <xs:import namespace="http://vsis-www.informatik.uni-hamburg.de/
         projects/management/logging.xsd"/>
5    <xs:element name="Steuerungsdokument">
6      <xs:annotation>
7        <xs:documentation>beschreibt den Aufbau des Steuerungsdokuments.<
             /xs:documentation>
8      </xs:annotation>
9      <xs:complexType>
10       <xs:sequence>
11         <xs:element name="Monitoring">
12           <xs:annotation>
13             <xs:documentation>beinhaltet Monitoring bezogene Parameter.
                   </xs:documentation>
14           </xs:annotation>
15           <xs:complexType>
16             <xs:sequence>
17               <xs:element name="Manageranzahl">
18                 <xs:annotation>
19                   <xs:documentation>legt fest, wie viele Manager den
                         Prozessbearbeiter unterstützen sollen.</
                         xs:documentation>
20                 </xs:annotation>
21                 <xs:simpleType>
22                   <xs:restriction base="xs:integer">
23                     <xs:minInclusive value="0"/>
```

```
24                  </xs:restriction>
25                </xs:simpleType>
26              </xs:element>
27              <xs:element name="MinManageranzahl" minOccurs="0">
28                <xs:annotation>
29                  <xs:documentation>gibt an, wie viele Manager den
                        Prozessbearbeiter wenigstens des Ausfalls
                        verdächtigen müssen, damit der Prozess
                        wiederhergestellt wird.</xs:documentation>
30                </xs:annotation>
31                <xs:simpleType>
32                  <xs:restriction base="xs:integer">
33                    <xs:minInclusive value="1"/>
34                  </xs:restriction>
35                </xs:simpleType>
36              </xs:element>
37              <xs:element name="Qualitätsparameter" minOccurs="0">
38                <xs:annotation>
39                  <xs:documentation>legt fest, nach welchen Kriterien
                        Manager gewählt werden (notwendig sofern
                        Manageranzahl >=1).</xs:documentation>
40                </xs:annotation>
41                <xs:simpleType>
42                  <xs:restriction base="xs:string">
43                    <xs:enumeration value="Batterielebensdauer"/>
44                    <xs:enumeration value="Rechenleistung"/>
45                  </xs:restriction>
46                </xs:simpleType>
47              </xs:element>
48              <xs:element name="Kommunikationswartezeit">
49                <xs:annotation>
50                  <xs:documentation>beschreibt in Millisekunden wie
                        lange auf eine Antwortnachricht eines anderen
                        Knotens gewartet wird (z. B. bei der Wahl eines
                        Managers auf die Antwort des Knotens).</
                        xs:documentation>
51                </xs:annotation>
52                <xs:simpleType>
53                  <xs:restriction base="xs:long">
54                    <xs:minInclusive value="0"/>
55                  </xs:restriction>
56                </xs:simpleType>
57              </xs:element>
58              <xs:element name="Heartbeatwartezeit" minOccurs="0">
59                <xs:annotation>
```

```
60                  <xs:documentation>legt die Wartezeit in Millisekunden
                         zwischen dem Senden zweier Heartbeats fest (
                        muss gesetzt werden wenn Manageranzahl >= 1).</
                        xs:documentation>
61                </xs:annotation>
62                <xs:simpleType>
63                  <xs:restriction base="xs:long">
64                    <xs:minInclusive value="0"/>
65                  </xs:restriction>
66                </xs:simpleType>
67              </xs:element>
68              <xs:element name="derzeitigeEndgeräte" minOccurs="0">
69                <xs:annotation>
70                  <xs:documentation>beinhaltet den Identifikator des
                        derzeitigen Prozessbearbeiters sowie eine Liste
                        aller derzeitigen Manager (muss gesetzt werden
                        wenn Manageranzahl >= 1).</xs:documentation>
71                </xs:annotation>
72                <xs:complexType>
73                  <xs:sequence>
74                    <xs:element name="ProzessbearbeiterID" type="
                           xs:string">
75                      <xs:annotation>
76                        <xs:documentation>Identifikator des derzeitigen
                               Prozessbearbeiters</xs:documentation>
77                      </xs:annotation>
78                    </xs:element>
79                    <xs:element name="ManagerID" type="xs:string"
                           maxOccurs="unbounded">
80                      <xs:annotation>
81                        <xs:documentation>Identifikator eines
                               derzeitigen Managers</xs:documentation>
82                      </xs:annotation>
83                    </xs:element>
84                  </xs:sequence>
85                </xs:complexType>
86              </xs:element>
87              <xs:any namespace="other" minOccurs="0" maxOccurs="
                    unbounded">
88                <xs:annotation>
89                  <xs:documentation>ermöglicht die Erweiterung des
                        Schemas.</xs:documentation>
90                </xs:annotation>
91              </xs:any>
92            </xs:sequence>
93          </xs:complexType>
```

```
94          </xs:element>
95          <xs:element name="Recovery">
96            <xs:annotation>
97              <xs:documentation>beinhaltet Recovery bezogene Parameter.</
                    xs:documentation>
98            </xs:annotation>
99            <xs:complexType>
100             <xs:sequence>
101               <xs:element ref="ns1:Kontrollpunktsynchronisation"
                      minOccurs="0"/>
102               <xs:element name="Kontrollpunkte" type="xs:boolean">
103                 <xs:annotation>
104                   <xs:documentation>legt fest, ob Kontrollpunkte
                          erstellt werden sollen.</xs:documentation>
105                 </xs:annotation>
106               </xs:element>
107               <xs:any namespace="other" minOccurs="0" maxOccurs="
                      unbounded">
108                 <xs:annotation>
109                   <xs:documentation>ermöglicht die Erweiterung des
                          Schemas.</xs:documentation>
110                 </xs:annotation>
111               </xs:any>
112             </xs:sequence>
113           </xs:complexType>
114         </xs:element>
115         <xs:element ref="ns2:Logging"/>
116         <xs:element name="Empfängerliste" minOccurs="0">
117           <xs:annotation>
118             <xs:documentation>optionale Liste von Empfängern der
                    Logdatei nach Abschluss bzw. während der Bearbeitung.<
                    /xs:documentation>
119           </xs:annotation>
120           <xs:complexType>
121             <xs:sequence>
122               <xs:element name="Ergebnisübermittlung">
123                 <xs:annotation>
124                   <xs:documentation>beschreibt zu welchen Zeitpunkten
                          Ergebnisse übermittelt werden.</xs:documentation
                          >
125                 </xs:annotation>
126                 <xs:complexType>
127                   <xs:choice>
128                     <xs:element name="Zeitpunkte">
129                       <xs:annotation>
```

```
130                     <xs:documentation>erlaubt es festzulegen, dass
                            nur am Ende der Prozessbearbeitung
                            Ergebnisse übermittelt werden oder exakt
                            zu den Zeitpunkten, wenn Kontrollpunkte
                            erstellt werden.</xs:documentation>
131                   </xs:annotation>
132                   <xs:simpleType>
133                     <xs:restriction base="xs:string">
134                       <xs:enumeration value="
                              Kontrollpunktsynchronisation"/>
135                       <xs:enumeration value="Ende"/>
136                     </xs:restriction>
137                   </xs:simpleType>
138                 </xs:element>
139                 <xs:any namespace="other" minOccurs="0" maxOccurs="
                        unbounded">
140                   <xs:annotation>
141                     <xs:documentation>ermöglicht die Erweiterung
                            des Schemas.</xs:documentation>
142                   </xs:annotation>
143                 </xs:any>
144               </xs:choice>
145             </xs:complexType>
146           </xs:element>
147           <xs:element name="EmpfängerID" type="xs:string" maxOccurs
                  ="unbounded">
148             <xs:annotation>
149               <xs:documentation>Identifikatoren der Empfänger der
                      Ergebnisse.</xs:documentation>
150             </xs:annotation>
151           </xs:element>
152           <xs:any namespace="other" minOccurs="0" maxOccurs="
                  unbounded">
153             <xs:annotation>
154               <xs:documentation>ermöglicht die Erweiterung des
                      Schemas.</xs:documentation>
155             </xs:annotation>
156           </xs:any>
157         </xs:sequence>
158         <xs:attribute name="Sendeversuche" type="xs:integer" use="
                required">
159           <xs:annotation>
160             <xs:documentation>legt fest, wie oft der
                    Prozessbearbeiter Ergebnisse an die Empfänger zu
                    übermitteln versucht.</xs:documentation>
161           </xs:annotation>
```

```
162           </xs:attribute>
163           <xs:attribute name="Wartezeit" type="xs:long" use="required
                  ">
164             <xs:annotation>
165               <xs:documentation>legt fest, wie lange zwischen zwei
                      Übermittlungsversuchen gewartet wird (in
                      Millisekunden).</xs:documentation>
166             </xs:annotation>
167           </xs:attribute>
168           <xs:attribute name="Rekrutenanzahl" use="required">
169             <xs:annotation>
170               <xs:documentation>legt fest, wie viele Rekruten
                      akquiriert werden sollen.</xs:documentation>
171             </xs:annotation>
172             <xs:simpleType>
173               <xs:restriction base="xs:integer">
174                 <xs:minInclusive value="0"/>
175               </xs:restriction>
176             </xs:simpleType>
177           </xs:attribute>
178           <xs:attribute name="Rekrutierungstiefe" use="required">
179             <xs:annotation>
180               <xs:documentation>legt fest, bis zu welcher Tiefe
                      Rekruten rekrutiert werden sollen.</
                      xs:documentation>
181             </xs:annotation>
182             <xs:simpleType>
183               <xs:restriction base="xs:integer">
184                 <xs:minInclusive value="0"/>
185               </xs:restriction>
186             </xs:simpleType>
187           </xs:attribute>
188         </xs:complexType>
189       </xs:element>
190       <xs:element name="autorisierteEndgeräte">
191         <xs:annotation>
192           <xs:documentation>beschreibt welche Endgeräte den Status
                  der Bearbeitung des Prozesses abfragen und den Abbruch
                   befehligen dürfen.</xs:documentation>
193         </xs:annotation>
194         <xs:complexType>
195           <xs:sequence>
196             <xs:element name="Abbruchliste" minOccurs="0">
197               <xs:annotation>
198                 <xs:documentation>stellt eine Liste von Endgeräten
                        dar, die den Abbruch der Bearbeitung befehligen
```

```
                    dürfen.</xs:documentation>
199               </xs:annotation>
200               <xs:complexType>
201                 <xs:sequence>
202                   <xs:element name="EndgerätID" type="xs:string"
                          maxOccurs="unbounded">
203                     <xs:annotation>
204                       <xs:documentation>Identifikatoren der Endgeräte
                              mit Abbruchkompetenz.</xs:documentation>
205                     </xs:annotation>
206                   </xs:element>
207                 </xs:sequence>
208               </xs:complexType>
209             </xs:element>
210             <xs:element name="Statusliste" minOccurs="0">
211               <xs:annotation>
212                 <xs:documentation>stellt eine Liste von Endgeräten
                        dar, die den Status der Bearbeitung abfragen
                        dürfen.</xs:documentation>
213               </xs:annotation>
214               <xs:complexType>
215                 <xs:sequence>
216                   <xs:element name="EndgerätID" type="xs:string"
                          maxOccurs="unbounded">
217                     <xs:annotation>
218                       <xs:documentation>Identifikatoren der Endgeräte
                              mit Statusabfragekompetenz.</
                             xs:documentation>
219                     </xs:annotation>
220                   </xs:element>
221                 </xs:sequence>
222               </xs:complexType>
223             </xs:element>
224             <xs:any namespace="other" minOccurs="0" maxOccurs="
                    unbounded">
225               <xs:annotation>
226                 <xs:documentation>ermöglicht die Erweiterung des
                        Schemas.</xs:documentation>
227               </xs:annotation>
228             </xs:any>
229           </xs:sequence>
230         </xs:complexType>
231       </xs:element>
232       <xs:any namespace="other" minOccurs="0" maxOccurs="unbounded">
233         <xs:annotation>
```

```
234             <xs:documentation>ermöglicht die Erweiterung des Schemas.</
                    xs:documentation>
235           </xs:annotation>
236         </xs:any>
237       </xs:sequence>
238       <xs:attribute name="ProzessID" type="xs:string" use="required">
239         <xs:annotation>
240           <xs:documentation>Identifikator des Prozesses für den
                  Management betrieben werden soll.</xs:documentation>
241         </xs:annotation>
242       </xs:attribute>
243     </xs:complexType>
244   </xs:element>
245 </xs:schema>
```

A.1.2 XML-Schema eines Kontrollpunktsynchronisations-Elements

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <xs:schema xmlns="http://vsis-www.informatik.uni-hamburg.de/projects/
      management/kontrollpunktsynchronisation.xsd" xmlns:xs="http://www.
      w3.org/2001/XMLSchema" targetNamespace="http://vsis-www.informatik
      .uni-hamburg.de/projects/management/kontrollpunktsynchronisation.
      xsd" elementFormDefault="qualified" attributeFormDefault="
      unqualified">
4   <xs:element name="Kontrollpunktsynchronisation">
5     <xs:annotation>
6       <xs:documentation>beschreibt wann Kontrollpunkte erstellt und mit
            den Managern synchronisiert werden.</xs:documentation>
7     </xs:annotation>
8     <xs:complexType>
9       <xs:choice>
10        <xs:element name="nachAktivität" type="xs:positiveInteger"
              minOccurs="0">
11          <xs:annotation>
12            <xs:documentation>beschreibt nach wievielen Aktivitäten
                  jeweils Kontrollpunkte erstellt werden. Beispiel: 3
                  besagt, dass nach jeder 3. Aktivität
                  Kontrollpunktsynchronisation stattfindet.</
                  xs:documentation>
13          </xs:annotation>
14        </xs:element>
15        <xs:element name="AktivitätsListe" minOccurs="0">
16          <xs:annotation>
```

```
17            <xs:documentation>stellt eine Liste von
                  Aktivitätsidentifikatoren dar, nach denen jeweils
                  Kontrollpunktsynchronisation stattfindet.</
                  xs:documentation>
18          </xs:annotation>
19          <xs:complexType>
20            <xs:sequence>
21              <xs:element name="AktivitätsID" type="xs:string"
                    maxOccurs="unbounded">
22                <xs:annotation>
23                  <xs:documentation>Identifikator einer Aktivität, nach
                        der Kontrollpunktsynchronisation durchgeführt
                        werden soll.</xs:documentation>
24                </xs:annotation>
25              </xs:element>
26            </xs:sequence>
27          </xs:complexType>
28        </xs:element>
29        <xs:any namespace="other" processContents="strict" minOccurs="0
              " maxOccurs="unbounded">
30          <xs:annotation>
31            <xs:documentation>ermöglicht die Erweiterung des Schemas.</
                  xs:documentation>
32          </xs:annotation>
33        </xs:any>
34      </xs:choice>
35    </xs:complexType>
36  </xs:element>
37 </xs:schema>
```

A.1.3 XML-Schema eines Logging-Elements

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns="http://vsis-www.informatik.uni-hamburg.de/projects/
       management/logging.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema
       " targetNamespace="http://vsis-www.informatik.uni-hamburg.de/
       projects/management/logging.xsd" elementFormDefault="qualified"
       attributeFormDefault="unqualified">
3    <xs:element name="Logging">
4      <xs:annotation>
5        <xs:documentation>beschreibt welche Informationen geloggt werden
             sollen.</xs:documentation>
6      </xs:annotation>
7      <xs:complexType>
8        <xs:sequence>
```

```
9         <xs:element name="Prozessbearbeiter" type="xs:boolean">
10          <xs:annotation>
11            <xs:documentation>Prozessbearbeiter loggen?</
                  xs:documentation>
12          </xs:annotation>
13        </xs:element>
14        <xs:element name="Manager" type="xs:boolean">
15          <xs:annotation>
16            <xs:documentation>Manager loggen?</xs:documentation>
17          </xs:annotation>
18        </xs:element>
19        <xs:element name="Dauer" type="xs:boolean">
20          <xs:annotation>
21            <xs:documentation>legt fest ob die Dauer der Aktivitäten
                  aufgezeichnet werden soll.</xs:documentation>
22          </xs:annotation>
23        </xs:element>
24        <xs:element name="Ergebnisse" type="xs:boolean">
25          <xs:annotation>
26            <xs:documentation>Ergebnisse speichern?</xs:documentation>
27          </xs:annotation>
28        </xs:element>
29        <xs:element name="Recoveryanzahl" type="xs:boolean">
30          <xs:annotation>
31            <xs:documentation>aufzeichnen wie oft der Prozess im Rahmen
                  der Recovery neugestartet wurde?</xs:documentation>
32          </xs:annotation>
33        </xs:element>
34        <xs:any namespace="other" minOccurs="0" maxOccurs="unbounded">
35          <xs:annotation>
36            <xs:documentation>ermöglicht die Erweiterung des Schemas um
                  beliebige weitere Parameter.</xs:documentation>
37          </xs:annotation>
38        </xs:any>
39      </xs:sequence>
40    </xs:complexType>
41  </xs:element>
42 </xs:schema>
```

A.1.4 Beispiel eines Steuerungsdokuments

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Steuerungsdokument ProzessID="PZ120" xsi:schemaLocation="http://vsis-
      www.informatik.uni-hamburg.de/projects/management/
      steuerungsdokument.xsd␣Steuerungsdokument.xsd" xmlns="http://vsis-
```

```
    www.informatik.uni-hamburg.de/projects/management/
    steuerungsdokument.xsd" xmlns:ns1="http://vsis-www.informatik.uni-
    hamburg.de/projects/management/kontrollpunktsynchronisation.xsd"
    xmlns:ns2="http://vsis-www.informatik.uni-hamburg.de/projects/
    management/logging.xsd" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance">
3     <Monitoring>
4     <Manageranzahl>2</Manageranzahl>
5     <MinManageranzahl>1</MinManageranzahl>
6     <Qualitätsparameter>Rechenleistung</Qualitätsparameter>
7     <Kommunikationswartezeit>8000</Kommunikationswartezeit>
8     <Heartbeatwartezeit>2000</Heartbeatwartezeit>
9     <derzeitigeEndgeräte>
10      <ProzessbearbeiterID>PB2</ProzessbearbeiterID>
11      <ManagerID>MA123</ManagerID>
12      <ManagerID>MA972</ManagerID>
13    </derzeitigeEndgeräte>
14  </Monitoring>
15  <Recovery>
16    <ns1:Kontrollpunktsynchronisation>
17      <ns1:nachAktivität>3</ns1:nachAktivität>
18    </ns1:Kontrollpunktsynchronisation>
19    <Kontrollpunkte>true</Kontrollpunkte>
20  </Recovery>
21  <ns2:Logging>
22    <ns2:Prozessbearbeiter>true</ns2:Prozessbearbeiter>
23    <ns2:Manager>true</ns2:Manager>
24    <ns2:Dauer>true</ns2:Dauer>
25    <ns2:Ergebnisse>true</ns2:Ergebnisse>
26    <ns2:Recoveryanzahl>true</ns2:Recoveryanzahl>
27  </ns2:Logging>
28  <Empfängerliste Wartezeit="10000" Rekrutenanzahl="2"
        Rekrutierungstiefe="0" Sendeversuche="3">
29    <Ergebnisübermittlung>
30      <Zeitpunkte>Ende</Zeitpunkte>
31    </Ergebnisübermittlung>
32    <EmpfängerID>Empf01</EmpfängerID>
33    <EmpfängerID>Empf02</EmpfängerID>
34  </Empfängerliste>
35  <autorisierteEndgeräte>
36    <Abbruchliste>
37      <EndgerätID>PB12</EndgerätID>
38      <EndgerätID>PB24</EndgerätID>
39    </Abbruchliste>
40    <Statusliste>
41      <EndgerätID>PB12</EndgerätID>
```

```
42        <EndgerätID>PB33</EndgerätID>
43      </Statusliste>
44    </autorisierteEndgeräte>
45 </Steuerungsdokument>
```

A.2 Logdatei

A.2.1 XML-Schema der Logdatei

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns="http://vsis-www.informatik.uni-hamburg.de/projects/
       management/logdateischema.xsd" xmlns:xs="http://www.w3.org/2001/
       XMLSchema" xmlns:ns1="http://vsis-www.informatik.uni-hamburg.de/
       projects/management/loginformationen.xsd" targetNamespace="http://
       vsis-www.informatik.uni-hamburg.de/projects/management/
       logdateischema.xsd" elementFormDefault="qualified"
       attributeFormDefault="unqualified">
3    <xs:import namespace="http://vsis-www.informatik.uni-hamburg.de/
         projects/management/loginformationen.xsd" schemaLocation="C:\
         Dokumente␣und␣Einstellungen\Xardas\Desktop\Diplomarbeit\Vorlagen
         \LaTeX␣Template\xml\loginformationen.xsd"/>
4    <xs:element name="Logdatei">
5      <xs:annotation>
6        <xs:documentation>beschreibt den strukturellen Aufbau einer
             Logdatei.</xs:documentation>
7      </xs:annotation>
8      <xs:complexType>
9        <xs:sequence>
10         <xs:element ref="ns1:Aktivitäts-Log" minOccurs="0"/>
11         <xs:element ref="ns1:Prozess-Log" minOccurs="0"/>
12       </xs:sequence>
13       <xs:attribute name="ProzessID" type="xs:string" use="required">
14         <xs:annotation>
15           <xs:documentation>Identifikator des Prozesses zu dem die
                 Logdatei gehört.</xs:documentation>
16         </xs:annotation>
17       </xs:attribute>
18     </xs:complexType>
19   </xs:element>
20 </xs:schema>
```

A.2.2 XML-Schema eines Aktivitäts-Log- und Prozess-Log Elements

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns="http://vsis-www.informatik.uni-hamburg.de/projects/
       management/kontrollpunktsynchronisation.xsd" xmlns:xs="http://www.
       w3.org/2001/XMLSchema" xmlns:ns1="http://vsis-www.informatik.uni-
       hamburg.de/projects/management/loginformationen.xsd"
       targetNamespace="http://vsis-www.informatik.uni-hamburg.de/
       projects/management/loginformationen.xsd" elementFormDefault="
       qualified" attributeFormDefault="unqualified">
3    <xs:element name="Aktivitäts-Log">
4      <xs:annotation>
5        <xs:documentation>beinhaltet sämtliche aktivittätsspezifische
             Loginformationen.</xs:documentation>
6      </xs:annotation>
7      <xs:complexType>
8        <xs:sequence>
9          <xs:element name="Aktivität" maxOccurs="unbounded">
10           <xs:complexType>
11             <xs:sequence>
12               <xs:element name="Dauer" type="xs:string" minOccurs="0">
13                 <xs:annotation>
14                   <xs:documentation>beschreibt die Zeitspanne, die zur
                         Bearbeitung der Aktivität benötigt wurde.</
                         xs:documentation>
15                 </xs:annotation>
16               </xs:element>
17               <xs:element name="Ergebnis" type="xs:string" minOccurs="0
                     " maxOccurs="unbounded">
18                 <xs:annotation>
19                   <xs:documentation>beinhaltet ein Ergebnis der
                         Aktivität.</xs:documentation>
20                 </xs:annotation>
21               </xs:element>
22               <xs:any namespace="other" minOccurs="0" maxOccurs="
                     unbounded">
23                 <xs:annotation>
24                   <xs:documentation>ermöglicht die Erweiterung des
                         Schemas um beliebige weitere Parameter.</
                         xs:documentation>
25                 </xs:annotation>
26               </xs:any>
27             </xs:sequence>
28             <xs:attribute name="AktivitätsID" type="xs:string" use="
                   required">
29               <xs:annotation>
```

```
30              <xs:documentation>eindeutiger Identifikator einer
                    Aktivität.</xs:documentation>
31            </xs:annotation>
32          </xs:attribute>
33          <xs:attribute name="Durchlauf" use="required">
34            <xs:annotation>
35              <xs:documentation>gibt an, der wievielte Durchlauf der
                    Aktivität mit diesem Identifikator gemeint ist (
                    Zur Unterscheidung von Schleifen-Durchläufen).</
                    xs:documentation>
36            </xs:annotation>
37            <xs:simpleType>
38              <xs:restriction base="xs:integer">
39                <xs:minInclusive value="1"/>
40              </xs:restriction>
41            </xs:simpleType>
42          </xs:attribute>
43        </xs:complexType>
44      </xs:element>
45    </xs:sequence>
46  </xs:complexType>
47 </xs:element>
48 <xs:element name="Prozess-Log">
49   <xs:annotation>
50     <xs:documentation>beinhaltet sämtliche prozessglobale
          Loginformationen</xs:documentation>
51   </xs:annotation>
52   <xs:complexType>
53     <xs:sequence>
54       <xs:element name="Prozessbearbeiterliste" minOccurs="0">
55         <xs:annotation>
56           <xs:documentation>beinhaltet Informationen über sämtliche
                Prozessbearbeiter sowie ggf. Informationen über die
                Manager und Aktivitäten. </xs:documentation>
57         </xs:annotation>
58         <xs:complexType>
59           <xs:sequence>
60             <xs:element name="Prozessbearbeiter" maxOccurs="unbounded
                  ">
61               <xs:annotation>
62                 <xs:documentation>beinhaltet Informationen über die
                      vom Prozessbearbeiter bearbeiteten Aktivitäten,
                      sowie darüber welche Manager ihn unterstützt
                      haben.</xs:documentation>
63               </xs:annotation>
64               <xs:complexType>
```

```
65                  <xs:sequence>
66                    <xs:element name="Fehler" type="xs:boolean"
                          minOccurs="0">
67                      <xs:annotation>
68                        <xs:documentation>gibt an, ob der Prozess
                              während der Bearbeitung durch einen
                              Manager wiederhergestellt werden musste.</
                              xs:documentation>
69                      </xs:annotation>
70                    </xs:element>
71                    <xs:element name="Aktivitätsreferenzen">
72                      <xs:complexType>
73                        <xs:sequence>
74                          <xs:element name="Aktivitätsreferenz"
                                minOccurs="0" maxOccurs="unbounded">
75                            <xs:annotation>
76                              <xs:documentation>erlaubt es nachträglich
                                    festzustellen, welcher
                                  Prozessbearbeiter welche
                                  Aktivitätsdurchläufe bearbeitet hat,
                                   indem für jeden Durchlauf einer
                                  Aktivität eine Aktivitätsreferenz
                                  beim jeweiligen Prozessbearbeiter
                                  erzeugt wird.</xs:documentation>
77                            </xs:annotation>
78                            <xs:complexType>
79                              <xs:sequence>
80                                <xs:element name="AktivitätsID" type="
                                      xs:string">
81                                  <xs:annotation>
82                                    <xs:documentation>eindeutiger
                                          Identifikator der Aktivität.</
                                          xs:documentation>
83                                  </xs:annotation>
84                                </xs:element>
85                                <xs:element name="Durchlauf">
86                                  <xs:annotation>
87                                    <xs:documentation>beschreibt der
                                          wievielte Durchlauf einer
                                          Aktivität gemeint ist (Zur
                                          Unterscheidung von Schleifen-
                                          Durchläufen).</
                                          xs:documentation>
88                                  </xs:annotation>
89                                  <xs:simpleType>
90                                    <xs:restriction base="xs:integer">
```

```
91                                  <xs:minInclusive value="1"/>
92                                </xs:restriction>
93                              </xs:simpleType>
94                            </xs:element>
95                          </xs:sequence>
96                        </xs:complexType>
97                      </xs:element>
98                    </xs:sequence>
99                  </xs:complexType>
100               </xs:element>
101               <xs:element name="Managerliste" minOccurs="0">
102                 <xs:annotation>
103                   <xs:documentation>beschreibt welche Manager den
                          Prozessbearbeiter in welcher Reihenfolge
                        unterstützt haben.</xs:documentation>
104                 </xs:annotation>
105                 <xs:complexType>
106                   <xs:sequence>
107                     <xs:element name="Startkonstellation">
108                       <xs:annotation>
109                         <xs:documentation>beschreibt die
                              Startkonstellation der Manger, die
                              den jeweiligen Prozessbearbeiter
                              unterstützen.</xs:documentation>
110                       </xs:annotation>
111                       <xs:complexType>
112                         <xs:sequence>
113                           <xs:element name="ManagerID" type="
                                xs:string" minOccurs="0" maxOccurs
                                ="unbounded">
114                             <xs:annotation>
115                               <xs:documentation>Anzahl der
                                    Manager-Identifikatoren wird
                                    durch den Parameter
                                    Mangeranzahl im
                                    Steuerungsdokument festgelegt.
                                    </xs:documentation>
116                             </xs:annotation>
117                           </xs:element>
118                         </xs:sequence>
119                       </xs:complexType>
120                     </xs:element>
121                     <xs:element name="Änderung" minOccurs="0"
                          maxOccurs="unbounded">
122                       <xs:annotation>
```

```
123                         <xs:documentation>verdeutlicht Änderungen
                                der Managerkonstellation.</
                               xs:documentation>
124                       </xs:annotation>
125                       <xs:complexType>
126                         <xs:sequence>
127                           <xs:element name="Zeitpunkt" minOccurs=
                                  "0">
128                             <xs:annotation>
129                               <xs:documentation>gibt an, wann der
                                      Manager aus der Managerliste
                                     entfernt/hinzugefügt wurde.</
                                     xs:documentation>
130                             </xs:annotation>
131                             <xs:complexType>
132                               <xs:sequence minOccurs="0">
133                                 <xs:element name="
                                       Aktivitätsreferenz"
                                       minOccurs="0">
134                                   <xs:annotation>
135                                     <xs:documentation>Dieser
                                          Eintrag ist nur relevant
                                          , wenn das Attribut
                                          relZeitpunkt mit"laufend
                                          " oder "beendet" belegt
                                          ist. Es gibt an auf
                                          welche Aktivität sich
                                          der relative Zeitpunkt
                                          der Änderung bezieht. </
                                          xs:documentation>
136                                   </xs:annotation>
137                                   <xs:complexType>
138                                     <xs:sequence>
139                                       <xs:element name="
                                            AktivitätsID" type="
                                            xs:string">
140                                         <xs:annotation>
141                                           <xs:documentation>
                                                eindeutiger
                                                Identifikator der
                                                Aktivität.</
                                                xs:documentation>
142                                         </xs:annotation>
143                                       </xs:element>
144                                       <xs:element name="Durchlauf
                                            " type="xs:integer">
```

```
145                                    <xs:annotation>
146                                      <xs:documentation>
                                             beschreibt der
                                             wievielte
                                             Durchlauf einer
                                             Aktivität gemeint
                                             ist (Zur
                                             Unterscheidung von
                                              Schleifen-
                                             Durchläufen).</
                                             xs:documentation>
147                                    </xs:annotation>
148                                  </xs:element>
149                                </xs:sequence>
150                              </xs:complexType>
151                            </xs:element>
152                            <xs:any namespace="other"
                                   minOccurs="0" maxOccurs="
                                   unbounded">
153                              <xs:annotation>
154                                <xs:documentation>ermöglicht
                                     die Erweiterung des
                                     Schemas um beliebige
                                     weitere Parameter.</
                                     xs:documentation>
155                              </xs:annotation>
156                            </xs:any>
157                          </xs:sequence>
158                          <xs:attribute name="relZeitpunkt">
159                            <xs:annotation>
160                              <xs:documentation>gibt an, zu
                                     welchem relativen
                                     Zeitpunkt der Manager
                                     hinzugefügt bzw. entfernt
                                     wurde. "start" bedeutet,
                                     dass der Manager vor der
                                     Bearbeitung der ersten
                                     Aktivität ersetzt/
                                     hinzugefügt wurde."laufend
                                     " besagt, dass die
                                     Änderung während der
                                     Bearbeitung einer
                                     Aktivität durchgeführt
                                     wurde und "beendet"
                                     bedeutet, dass die
                                     Änderung nach der
```

```
                                    Bearbeitung einer
                                    Aktivität durchgeführt
                                    wurde. Auf welche
                                    Aktivität bezug genommen
                                    wird, wird durch die
                                    Aktiviitätsreferenz
                                    festgelegt.</
                                    xs:documentation>
161                             </xs:annotation>
162                             <xs:simpleType>
163                               <xs:restriction base="xs:string
                                      ">
164                                 <xs:enumeration value="start"
                                        />
165                                 <xs:enumeration value="
                                        laufend"/>
166                                 <xs:enumeration value="
                                        beendet"/>
167                               </xs:restriction>
168                             </xs:simpleType>
169                           </xs:attribute>
170                         </xs:complexType>
171                       </xs:element>
172                     </xs:sequence>
173                     <xs:attribute name="ManagerID" type="
                          xs:string">
174                       <xs:annotation>
175                         <xs:documentation>eindeutiger
                              Identifikator des Managers.</
                              xs:documentation>
176                       </xs:annotation>
177                     </xs:attribute>
178                     <xs:attribute name="Art">
179                       <xs:annotation>
180                         <xs:documentation>gibt an, ob es sich
                               bei der Änderung um die
                              Entfernung eines bestehenden
                              oder das Hinzufügen eines neuen
                              Managers handelt.</
                              xs:documentation>
181                       </xs:annotation>
182                       <xs:simpleType>
183                         <xs:restriction base="xs:string">
184                           <xs:enumeration value="
                                ManagerEntfernung"/>
```

```
185                                    <xs:enumeration value="neuerManager
                                           "/>
186                                  </xs:restriction>
187                                </xs:simpleType>
188                              </xs:attribute>
189                            </xs:complexType>
190                          </xs:element>
191                        </xs:sequence>
192                      </xs:complexType>
193                    </xs:element>
194                    <xs:any namespace="other" minOccurs="0" maxOccurs="
                           unbounded">
195                      <xs:annotation>
196                        <xs:documentation>ermöglicht die Erweiterung
                               des Schemas um beliebige weitere Parameter
                               .</xs:documentation>
197                      </xs:annotation>
198                    </xs:any>
199                  </xs:sequence>
200                  <xs:attribute name="ProzessbearbeiterID" type="
                         xs:string" use="required">
201                    <xs:annotation>
202                      <xs:documentation>eindeutiger Identifikator des
                             Prozessbearbeiters.</xs:documentation>
203                    </xs:annotation>
204                  </xs:attribute>
205                </xs:complexType>
206              </xs:element>
207            </xs:sequence>
208          </xs:complexType>
209        </xs:element>
210        <xs:element name="Managerliste" minOccurs="0">
211          <xs:annotation>
212            <xs:documentation>beinhaltet Informationen über sämtliche
                   Manager, die an der Bearbeitung des Prozesses
                   beteiligt waren. Dieses Element sollte nur dann
                   verwendet werden, wenn im zugehörigen
                   Steuerungsdokument festgelegt wurde, dass keine
                   Prozessbearbeiter, sondern nur die Manager gespeichert
                    werden sollen.</xs:documentation>
213          </xs:annotation>
214          <xs:complexType>
215            <xs:sequence>
216              <xs:element name="Änderung" minOccurs="0" maxOccurs="
                     unbounded">
217                <xs:annotation>
```

```
218                 <xs:documentation>verdeutlicht Änderungen der
                        Managerkonstellation.</xs:documentation>
219               </xs:annotation>
220               <xs:complexType>
221                 <xs:sequence>
222                   <xs:element name="Zeitpunkt" minOccurs="0">
223                     <xs:annotation>
224                       <xs:documentation>gibt an wann der Manager aus
                            der Managerliste entfernt/hinzugefügt
                            wurde.</xs:documentation>
225                     </xs:annotation>
226                     <xs:complexType>
227                       <xs:sequence minOccurs="0">
228                         <xs:element name="Aktivitätsreferenz"
                              minOccurs="0">
229                           <xs:annotation>
230                             <xs:documentation>Dieser Eintrag ist nur
                                  relevant, wenn das Attribut
                                  relZeitpunkt mit"laufend" oder "
                                  beendet" belegt ist. Es gibt an auf
                                  welche Aktivität sich der relative
                                  Zeitpunkt der Änderung bezieht. </
                                  xs:documentation>
231                           </xs:annotation>
232                           <xs:complexType>
233                             <xs:sequence>
234                               <xs:element name="AktivitätsID" type="
                                    xs:string">
235                                 <xs:annotation>
236                                   <xs:documentation>eindeutiger
                                        Identifikator der Aktivität.</
                                        xs:documentation>
237                                 </xs:annotation>
238                               </xs:element>
239                               <xs:element name="Durchlauf" type="
                                    xs:integer">
240                                 <xs:annotation>
241                                   <xs:documentation>beschreibt der
                                        wievielte Durchlauf einer
                                        Aktivität gemeint ist (Zur
                                        Unterscheidung von Schleifen-
                                        Durchläufen).</
                                        xs:documentation>
242                                 </xs:annotation>
243                               </xs:element>
244                             </xs:sequence>
```

```
245                         </xs:complexType>
246                       </xs:element>
247                       <xs:any namespace="other" minOccurs="0"
                              maxOccurs="unbounded">
248                         <xs:annotation>
249                           <xs:documentation>ermöglicht die
                                Erweiterung des Schemas um beliebige
                                 weitere Parameter.</
                                xs:documentation>
250                         </xs:annotation>
251                       </xs:any>
252                     </xs:sequence>
253                     <xs:attribute name="relZeitpunkt">
254                       <xs:annotation>
255                         <xs:documentation>gibt an, zu welchem
                              relativen Zeitpunkt der Manager
                              hinzugefügt bzw. entfernt wurde. "
                              start" bedeutet, dass der Manager vor
                              der Bearbeitung der ersten Aktivität
                              ersetzt/hinzugefügt wurde."laufend"
                              besagt, dass die Änderung während der
                              Bearbeitung einer Aktivität
                              durchgeführt wurde und "beendet"
                              bedeutet, dass die Änderung nach der
                              Bearbeitung einer Aktivität
                              durchgeführt wurde. Auf welche
                              Aktivität bezug genommen wird, wird
                              durch die Aktiviitätsreferenz
                              festgelegt.</xs:documentation>
256                       </xs:annotation>
257                       <xs:simpleType>
258                         <xs:restriction base="xs:string">
259                           <xs:enumeration value="start"/>
260                           <xs:enumeration value="laufend"/>
261                           <xs:enumeration value="beendet"/>
262                         </xs:restriction>
263                       </xs:simpleType>
264                     </xs:attribute>
265                   </xs:complexType>
266                 </xs:element>
267               </xs:sequence>
268               <xs:attribute name="ManagerID" type="xs:string">
269                 <xs:annotation>
270                   <xs:documentation>eindeutiger Identifikator des
                        Managers.</xs:documentation>
271                 </xs:annotation>
```

```
272                 </xs:attribute>
273                 <xs:attribute name="Art">
274                   <xs:annotation>
275                     <xs:documentation>gibt an, ob es sich bei der
                            Änderung um die Entfernung eines bestehenden
                             oder das Hinzufügen eines neuen Managers
                            handelt.</xs:documentation>
276                   </xs:annotation>
277                   <xs:simpleType>
278                     <xs:restriction base="xs:string">
279                       <xs:enumeration value="ManagerEntfernung"/>
280                       <xs:enumeration value="neuerManager"/>
281                     </xs:restriction>
282                   </xs:simpleType>
283                 </xs:attribute>
284               </xs:complexType>
285             </xs:element>
286           </xs:sequence>
287         </xs:complexType>
288       </xs:element>
289       <xs:element name="Anfragen">
290         <xs:complexType>
291           <xs:sequence>
292             <xs:element name="Anfrage" minOccurs="0" maxOccurs="
                    unbounded">
293               <xs:complexType>
294                 <xs:sequence>
295                   <xs:element name="AnfrageID" type="xs:string"/>
296                   <xs:element name="Anfrager" type="xs:string"/>
297                 </xs:sequence>
298                 <xs:attribute name="Art">
299                   <xs:simpleType>
300                     <xs:restriction base="xs:string">
301                       <xs:enumeration value="Status"/>
302                       <xs:enumeration value="Abbruch"/>
303                       <xs:enumeration value="Suche"/>
304                     </xs:restriction>
305                   </xs:simpleType>
306                 </xs:attribute>
307                 <xs:attribute name="Erlaubt" type="xs:boolean"/>
308               </xs:complexType>
309             </xs:element>
310           </xs:sequence>
311         </xs:complexType>
312       </xs:element>
313       <xs:element name="Recoveryanzahl" minOccurs="0">
```

```
314           <xs:annotation>
315             <xs:documentation>gibt an, wie oft Recovery durchgeführt
                    wurde.</xs:documentation>
316           </xs:annotation>
317           <xs:simpleType>
318             <xs:restriction base="xs:integer">
319               <xs:minInclusive value="0"/>
320             </xs:restriction>
321           </xs:simpleType>
322         </xs:element>
323         <xs:any namespace="other" minOccurs="0" maxOccurs="unbounded">
324           <xs:annotation>
325             <xs:documentation>ermöglicht die Erweiterung des Schemas um
                    beliebige weitere Parameter.</xs:documentation>
326           </xs:annotation>
327         </xs:any>
328       </xs:sequence>
329     </xs:complexType>
330   </xs:element>
331 </xs:schema>
```

A.2.3 Beispiel einer Logdatei

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Logdatei ProzessID="PZ120" xsi:schemaLocation="http://vsis-www.
       informatik.uni-hamburg.de/projects/management/logdateischema.xsd␣
       logdateischema.xsd" xmlns="http://vsis-www.informatik.uni-hamburg.
       de/projects/management/logdateischema.xsd" xmlns:ns1="http://vsis-
       www.informatik.uni-hamburg.de/projects/management/loginformationen
       .xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3    <ns1:Aktivitäts-Log>
4      <ns1:Aktivität Durchlauf="1" AktivitätsID="AKT123">
5        <ns1:Dauer>0:0:23:7</ns1:Dauer>
6        <ns1:Ergebnis>55</ns1:Ergebnis>
7      </ns1:Aktivität>
8      <ns1:Aktivität Durchlauf="1" AktivitätsID="AKT521">
9        <ns1:Dauer>0:2:10:30</ns1:Dauer>
10       <ns1:Ergebnis>99</ns1:Ergebnis>
11     </ns1:Aktivität>
12   </ns1:Aktivitäts-Log>
13   <ns1:Prozess-Log>
14     <ns1:Prozessbearbeiterliste>
15       <ns1:Prozessbearbeiter ProzessbearbeiterID="PB1">
16         <ns1:Fehler>false</ns1:Fehler>
17         <ns1:Aktivitätsreferenzen>
```

```
18          <ns1:Aktivitätsreferenz>
19            <ns1:AktivitätsID>AKT123</ns1:AktivitätsID>
20            <ns1:Durchlauf>1</ns1:Durchlauf>
21          </ns1:Aktivitätsreferenz>
22        </ns1:Aktivitätsreferenzen>
23        <ns1:Managerliste>
24          <ns1:Startkonstellation>
25          </ns1:Startkonstellation>
26          <ns1:Änderung Art="neuerManager" ManagerID="MA5">
27            <ns1:Zeitpunkt relZeitpunkt="start">
28            </ns1:Zeitpunkt>
29          </ns1:Änderung>
30          <ns1:Änderung Art="neuerManager" ManagerID="MA123">
31            <ns1:Zeitpunkt relZeitpunkt="start">
32            </ns1:Zeitpunkt>
33          </ns1:Änderung>
34        </ns1:Managerliste>
35      </ns1:Prozessbearbeiter>
36      <ns1:Prozessbearbeiter ProzessbearbeiterID="PB2">
37        <ns1:Fehler>false</ns1:Fehler>
38        <ns1:Aktivitätsreferenzen>
39          <ns1:Aktivitätsreferenz>
40            <ns1:AktivitätsID>AKT521</ns1:AktivitätsID>
41            <ns1:Durchlauf>1</ns1:Durchlauf>
42          </ns1:Aktivitätsreferenz>
43        </ns1:Aktivitätsreferenzen>
44        <ns1:Managerliste>
45          <ns1:Startkonstellation>
46            <ns1:ManagerID>MA5</ns1:ManagerID>
47            <ns1:ManagerID>MA123</ns1:ManagerID>
48          </ns1:Startkonstellation>
49          <ns1:Änderung Art="ManagerEntfernung" ManagerID="MA123">
50            <ns1:Zeitpunkt relZeitpunkt="laufend">
51              <ns1:Aktivitätsreferenz>
52                <ns1:AktivitätsID>AKT521</ns1:AktivitätsID>
53                <ns1:Durchlauf>1</ns1:Durchlauf>
54              </ns1:Aktivitätsreferenz>
55            </ns1:Zeitpunkt>
56          </ns1:Änderung>
57          <ns1:Änderung Art="neuerManager" ManagerID="MA972">
58            <ns1:Zeitpunkt relZeitpunkt="laufend">
59              <ns1:Aktivitätsreferenz>
60                <ns1:AktivitätsID>AKT521</ns1:AktivitätsID>
61                <ns1:Durchlauf>1</ns1:Durchlauf>
62              </ns1:Aktivitätsreferenz>
63            </ns1:Zeitpunkt>
```

```
64              </ns1:Änderung>
65            </ns1:Managerliste>
66          </ns1:Prozessbearbeiter>
67        </ns1:Prozessbearbeiterliste>
68        <ns1:Anfragen>
69          <ns1:Anfrage Art="Abbruch" Erlaubt="false">
70            <ns1:AnfrageID>Anf12123</ns1:AnfrageID>
71            <ns1:Anfrager>PB1232</ns1:Anfrager>
72          </ns1:Anfrage>
73        </ns1:Anfragen>
74        <ns1:Recoveryanzahl>0</ns1:Recoveryanzahl>
75    </ns1:Prozess-Log>
76 </Logdatei>
```

A.3 Pseudocode des verwendeten adaptiven Heartbeat-Algorithmus

Initialisierung:

Sei v ein Knoten der Heartbeats an u sendet und p der zugehörige Prozess.

$lokalerHeartbeat - Counter_{v,p} = 0;$

$lokHops_{v,p} = 0;$

$\alpha_0^{v,p} = var_0^{v,p} = error_0^{v,p} = 0;$

$A_0^{v,p} = var_0^{v,p} = error_0^{v,p} = 0;$

$\tau_0^{v,p} = EA_0^{v,p} = U_0^{v,p} = 0;$

```
1  //Heartbeats senden. Für jeden durch Manager geschützten Prozess p
       bezüglich dessen ein Knoten v als Prozessbearbeiter oder Manager
       arbeitet gilt:
2  Task SendHeartbeats(){
3    while(true) {
4      Heartbeat - Counter_p ++;
5      Send(<"I am alive", v, p, Empfängerliste, Heartbeat-Counter_p, 0>);
6      //bis zum nächsten Heartbeat warten
7      Wait(Δ_p);
8      }
9  }
10 //Heartbeats verarbeiten. Verarbeitung eines empfangenen Heartbeats vom
       Sender v, bezüglich eines Prozesses p.
11 Task ProcessHeartbeats(){
12   while(true) {
13     Receive(m_v=<"I am alive", v, p, Empfängerliste, Heartbeat-Counter,
           Hops>);
14
```

```
15      if(u ∉ Empfängerliste) {
16        Send(<"I␣am␣alive", v, p, Empfängerliste, Heartbeat−Counter, Hops
              ++ >);
17        }
18      else {
19        //Nachricht nur weiterleiten, wenn u nicht der einzige in der
              Empfängerliste ist.
20        if(∃x  x ≠ u  x ∈ Empfängerliste ){
21          Send(<"I␣am␣alive", v, p, Empfängerliste, Heartbeat−Counter,
                Hops++ >);
22          }
23        t = LocalTime();
24
25        if(Heartbeat − Counter > lokalerHeartbeat − Counter_{v,p}){
26          if{Hops == 0}{
27            computeForNeighbors(v, p, Heartbeat − Counter, t);
28            }
29          else{
30            computeForNonNeighbors(v, p, Hops_{v,p});
31              }
32          lokalerHeartbeat − Counter_{v,p} = Heartbeat − Counter;
33          StartTimer(τ^{v,p}_{k+1});
34          }
35        //Sofern ein alter Heartbeat über einen längeren Weg ankommt,
              Wartezeit anpassen.
36        else if(Heartbeat − Counter == lokalerHeartbeat − Counter_{v,p}){
37          if(Hops ≠ 0){
38            lokHops_{v,p} = Hops;
39            x_{v,p} = EstHops_{v,p};
40            EstHops_{v,p} = EstimateGap(lokHops_{v,p});
41            if(EstHops_{v,p} > x_{v,p}){
42              τ^{v,p}_{k+1} = τ^{v,p}_{k+1} + (EstHops_{v,p} − x_{v,p}) · (δ + β);
43            }
44          }
45        }
46      }
47    }
48  }
49
50  //Berechnung für benachbarte Knoten
51  Procedure computeForNeighbors(v, p, k_{v,p}, t){
52    error^{v,p}_k = t − EA^{v,p}_k − delay^{v,p}_k;
53    delay^{v,p}_{k+1} = delay^{v,p}_k + γ · error^{v,p}_k;
54    var^{v,p}_{k+1} = var^{v,p}_k + γ · (|error^{v,p}_k| − var^{v,p}_k);
55    α^{v,p}_{k+1} = β · delay^{v,p}_{k+1} + φ · var^{v,p}_{k+1};
56
```

```
57    if ($k_{v,p} < n$) {
58       $U^{v,p}_{k+1} = \frac{t+k \cdot U_k}{k+1}$;
59       $EA^{v,p}_{k+1} = U_{k+1} + \frac{k+1}{2} \cdot \Delta$;
60       }
61    else{
62       $EA^{v,p}_{k+1} = EA^{v,p}_{k} + \frac{1}{n}(A^{v,p}_{k} - A^{v,p}_{k-n-1})$;
63       $A^{v,p}_{k} = t$;
64       $\tau^{v,p}_{k+1} = EA^{v,p}_{k+1} + \alpha^{v,p}_{k+1}$;
65       }
66 }
67 //Berechnung für nicht benachbarte Knoten
68 Procedure computeForNonNeighbors(v, p, $Hops_{v,p}$){
69    $EstHops_{v,p} = EstimateGap(Hops_{v,p})$;
70    $\tau^{v,p}_{k+1} = (\Delta_{v,p} + \delta) + EstHops_{v,p} \cdot (\delta + \beta)$;
71 }
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ADB+99] ABOWD, Gregory D. ; DEY, Anind K. ; BROWN, Peter J. ; DAVIES, Nigel ; SMITH, Mark ; STEGGLES, Pete: Towards a Better Understanding of Context and Context-Awareness. In: *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK : Springer-Verlag, 1999. – ISBN 3540665501, S. 304–307

[All07] ALLWEYER, Thomas.: *Geschäftsprozessmanagement : Strategie, Entwurf, Implementierung, Controlling*. 2. Nachdr.. Herdecke [u.a.] : W3L-Verl., 2007. – X, 427 S. : Ill., zahlr. graph. Darst.. – ISBN 978–3–937137–11–7

[AW01] AUBERLE, Anette ; WERMKE, Matthias: *Herkunftswörterbuch : Etymologie der deutschen Sprache ; auf der Grundlage der neuen amtlichen Rechtschreibregeln*. 3., völlig neu bearb. und erw. Aufl. Mannheim [u.a.] : Dudenverl., 2001 (Der Duden : in zwölf Bänden; das Standardwerk zur deutschen Sprache / hrsg. vom Wiss. Rat der Dudenred.: Matthias Wermke ... ; Bd. 7). – 960 S.. – ISBN 3411040734

[Bal00] BALZERT, Helmut: *Lehrbücher der Informatik*. Bd. 1. Software-Entwicklung: *Lehrbuch der Software-Technik*. 2. Aufl. Heidelberg [u.a.] : Spektrum, Akad.-Verl., 2000. – XX, 1136 S. : Ill., graph. Darst. + 2 CD–ROM. – ISBN 3–8274–0480–0

[Ben04] BENGEL, Günther: *Grundkurs verteilte Systeme : Grundlagen und Praxis des Client-Server-Computing - inklusive aktueller Technologien wie Web-Services u. a. ; für Studenten und Praktiker*. 3., verb. und erw. Aufl. Wiesbaden : Vieweg, 2004. – XV, 443 S. : graph. Darst.. – ISBN 3–528–25738–5

[BLP04] BREISIG, Thomas ; LOEBER-PAUTSCH, Uta: *Management*. 2004 (Berufsbegleitender internetgestützter Bachelor-Studiengang Business Administration (BA) in kleinen und mittleren Unternehmen)

[BMS02] BERTIER, Marin ; MARIN, Olivier ; SENS, Pierre: Implementation and Performance Evaluation of an Adaptable Failure Detector. In: *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1597–5, S. 354–363

[CDK05] COULOURIS, George F. ; DOLLIMORE, Jean ; KINDBERG, Tim: *Distributed systems : concepts and design*. 4. ed. Harlow [u.a.] : Addison-Wesley, 2005 (International computer science series). – XIV, 927 S. : Ill., graph. Darst.. – ISBN 0–321–26354–5

[CT96] CHANDRA, Tushar D. ; TOUEG, Sam: Unreliable Failure Detectors for Reliable Distributed Systems. In: *Journal of the ACM* 43 (1996), S. 225–267

[CTA00] CHEN, Wei ; TOUEG, Sam ; AGUILERA, Marcos K.: On the Quality of Service of Failure Detectors. In: *IEEE Transactions on Computers* 51 (2000), S. 561–580

[EAN07] ELHADEF, Mourad ; ABDUN-NUR, Fahim: An Adaptable Crash Faults Detector for Mobile Ad-Hoc Networks. In: *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2847–3, S. 207–212

[EB07] ELHADEF, Mourad ; BOUKERCHE, Azzedine: A Gossip-Style Crash Faults Detection Protocol for Wireless Ad-Hoc and Mesh Networks. In: *IPCCC*, IEEE Computer Society, 2007, S. 600–605

[Emm03] EMMERICH, Wolfgang: *Konstruktion von verteilten Objekten*. Heidelberg : dpunkt-Verl., 2003. – 501 S. : Ill., 25 cm. – ISBN 3–89864–140–6

[Eri01] ERICSSON (Hrsg.): *Von GSM zu UMTS*. Version: 2001. `http://www.ericsson.com/de/broschueren/von_gsm_zu_umts.pdf`, Abruf: 13. Januar 2009

[FC87] FINLAYSON, R. ; CHERITON, D.: Log files: an extended file service exploiting write-once storage. In: *SIGOPS Oper. Syst. Rev.* 21 (1987), Nr. 5, S. 139–148. `http://dx.doi.org/http://doi.acm.org/10.1145/37499.37516`. – DOI http://doi.acm.org/10.1145/37499.37516. – ISSN 0163–5980

[FPV98] FUGGETTA, A. ; PICCO, G. P. ; VIGNA, G.: Understanding code mobility. In: *Software Engineering, IEEE Transactions on* 24 (1998), Nr. 5, S. 342–361

[FT05] FRIEDMAN, Roy ; TCHARNY, Galaya: Evaluating Failure Detection in Mobile Ad-Hoc Networks. In: *International Journal of Wireless and Mobile Computing* Bd. 1, 2005

[FZ94] FORMAN, George H. ; ZAHORJAN, John: The Challenges of Mobile Computing. In: *Computer* 27 (1994), April, Nr. 4, S. 38–47

[Gad03] GADATSCH, Andreas: *Grundkurs Geschäftsprozess-Management : Methoden und Werkzeuge für die IT-Praxis ; eine Einführung für Studenten und Praktiker*. 3., verb. und erw. Aufl. Wiesbaden : Vieweg, 2003. – XXIII, 455 S. : Ill., graph. Darst.. – ISBN 3–528–25759–8

[Gad08] GADATSCH, Andreas: *Grundkurs Geschäftsprozess-Management : Methoden und Werkzeuge für die IT-Praxis ; eine Einführung für Studenten und Praktiker ; [mit*

Online-Service zum Buch]. 5., erw. und Überarbeitete Aufl. Wiesbaden : Vieweg, 2008. – XXIV, 480 S. : Ill., zahlr. graph. Darst., 24 cm. – ISBN 978–3–8348–0363–4

[GHS95] GEORGAKOPOULOS, Dimitrios ; HORNICK, Mark F. ; SHETH, Amit P.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. In: *Distributed and Parallel Databases* 3 (1995), Nr. 2, S. 119–153

[Gün04] GÜNE¸S, Mesut: *Routing und Adressierung in mobilen multi-hop Ad-hoc-Netzen*. Aachen, Germany, RWTH Aachen Universtiy, Diss., 2004

[GR93] GRAY, Jim ; REUTER, Andreas: *Transaction processing : concepts and techniques*. Corr. 2. print. Morgan Kaufmann Publ., 1993 (The Morgan Kaufmann series in data management systems). – XXXII; 1070 S. : graph. Darst.. – ISBN 1–55860–190–2 74,95

[Ham05] HAMMERSCHALL, Ulrike: *Verteilte Systeme und Anwendungen : Architekturkonzepte, Standards und Middleware-Technologien*. München [u.a.] : Pearson Studium, 2005. – 208 S. : graph. Darst.. – ISBN 3–8273–7096–5

[Han03] HANSMANN, Uwe: *Pervasive computing : the mobile world*. 2. ed. Berlin, Heidelberg , New York , Hongkong , London , Mailand , P Paris , Tokio : Springer, 2003. – XX, 448 S. : Ill., graph. Darst., 25 cm. – ISBN 3–540–00218–9

[HC94] HAMMER, Michael ; CHAMPY, James: *Business reengineering : die Radikalkur für das Unternehmen*. 2. Aufl. Frankfurt/Main, New York : Campus-Verl., 1994. – 288 S., 23 cm. – ISBN 3–593–35017–3 ; Gb.

[HD05] HAUSWIRTH, Manfred ; DUSTDAR, Schahram: Peer-to-Peer: Grundlagen und Architektur. In: *Datenbank-Spektrum* 13 (2005), S. 5–13

[Heg05] HEGERING, Heinz-Gerd: Management-Herausforderungen bei Grids. In: *Wissenschaftsmanagement* 1 (2005), S. 8–9

[Hes05] HESS, Thomas: Technische Möglichkeiten und Akzeptanz mobiler Anwendungen - eine interdisziplinäre Betrachtung. In: *Wirtschaftsinformatik* (2005), Nr. 1, S. 6–16. – ISSN 0937–6429

[Int92] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (Hrsg.): *Basic Reference Model of Open Distributed Processing, Part1: Overview and Guide to use*. ISO/IEC JTC1/SC212/WG7 CD 10746-1. International Organization for Standardization, 1992

[Jab97] JABLONSKI, Stefan: *Workflow-Management : Entwicklung von Anwendungen und Systemen ; Facetten einer neuen Technologie*. 1. Aufl. Heidelberg : dpunkt-Verl.,

1997 (dpunkt-Lehrbuch). – XII, 537 S. : graph. Darst., 24 cm. – ISBN 3–920993–73–X

[JLH+99] JOHANSSON, Per ; LARSSON, Tony ; HEDMAN, Nicklas ; MIELCZAREK, Bartosz ; DEGERMARK, Mikael: Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In: *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, ACM, 1999. – ISBN 1–58113–142–9, S. 195–206

[JLSU87] JOYCE, Jeffrey ; LOMOW, Greg ; SLIND, Konrad ; UNGER, Brian: Monitoring distributed systems. In: *ACM Trans. Comput. Syst.* 5 (1987), Nr. 2, S. 121–150. – ISSN 0734–2071

[Kel99] KELLER, Alexander: *CORBA-basiertes Enterprise-Management: Interoperabilität und Managementinstrumentierung verteilter kooperativer Managementsysteme in heterogener Umgebung*. München : Utz, Wiss., 1999 (Informatik). – IV, 306 S : Ill., graph. Darst ; 21 cm. S. – ISBN 3896754955 ((brosch.))

[Kle95] KLEINROCK, Leonard: *Nomadic computing*. 1995

[Kle96] KLEINROCK, Leonard: Nomadicity: anytime, anywhere in a disconnected world. In: *Mob. Netw. Appl.* 1 (1996), December, Nr. 4, S. 351–357

[KND98] KOKSAL, Pinar ; NURAL, Sena ; DOGAC, Arpinar A.: Workflow history management. In: *ACM Sigmod Record* 27 (1998), S. 67–75

[Kun05] KUNZE, Christian P.: DEMAC: A Distributed Environment for Mobility-Aware Computing. In: FERSCHA, A. (Hrsg.) ; MAYRHOFER, R. (Hrsg.) ; STRANG, T. (Hrsg.) ; LINNHOFF-POPIEN, C. (Hrsg.) ; DEY, A. (Hrsg.) ; BUTZ, A. (Hrsg.) ; A., Schmidt (Hrsg.): *Adjunct Proceedings of the Third International Conference on Pervasive Computing*, Oesterreichische Computer Gesellschaft, 5 2005, S. 115–121

[Kun08] KUNZE, Christian P.: *Kontextbasierte Kooperation: Unterstützung verteilter Prozesse im Mobile Computing*. Hamburg, Germany, Universität Hamburg, Diss., 2008

[KZL06] KUNZE, Christian P. ; ZAPLATA, Sonja ; LAMERSDORF, Winfried: Mobile Process Description and Execution. In: *DAIS*, 2006, S. 32–47

[KZL07a] KUNZE, Christian P. ; ZAPLATA, Sonja ; LAMERSDORF, Winfried: Abstrakte Dienstklassen zur Realisierung mobiler Prozesse. In: BRAUN, Torsten (Hrsg.) ; CARLE, Georg (Hrsg.) ; STILLER, Burkhard (Hrsg.) ; Gesellschaft für Informatik (Veranst.): *Konferenzband zur KiVS 2007 für Industrie-, Kurz- und Workshopbeiträge* Gesellschaft für Informatik, VDE Verlag, 2 2007, S. 123 – 128

[KZL07b] KUNZE, Christian P. ; ZAPLATA, Sonja ; LAMERSDORF, Winfried: Mobile Processes: Enhancing Cooperation in Distributed Mobile Environments. In: *Journal of Computers* 2 (2007), 2, Nr. 1, S. 1–11. – ISSN : 1796-203X

[KZTL08] KUNZE, Christian P. ; ZAPLATA, Sonja ; TURJALEI, Mirwais ; LAMERSDORF, Winfried: Enabling Context-based Cooperation: A Generic Context Model and Management System. In: ABRAMOWICZ, Witold (Hrsg.) ; FENSEL, Dieter (Hrsg.): *Business Information Systems*, Springer, 5 2008. – ISBN: 978-3-540-79395-3

[LM03] LANGHEINRICH, Marc ; MATTERN, Friedemann: Digitalisierung des Alltags. Was ist Pervasive Computing? In: *Politik und Zeitgeschichte (B 42/2003)* (2003), Oktober, S. 6–12

[LR00] LEYMANN, Frank ; ROLLER, Dieter: *Production workflow : concepts and techniques*. Upper Saddle River, NJ [u.a.] : Prentice Hall, 2000. – XXVIII, 479 S. : Ill., graph. Darst.. – ISBN 0–13–021753–0

[LY02] LYYTINEN, Kalle ; YOO, Youngjin: Issues and challenges in ubiquitous computing. In: *Commun. ACM* 45 (2002), Nr. 12

[Mat03] MATTERN, Friedemann: *Vom Verschwinden des Computers - die Vision des Ubiquitous Computing*. Berlin [u.a.], 2003

[MCA06] MORAIS CORDEIRO, Carlos d. ; AGRAWAL, Dharma P.: *Ad hoc and sensor networks : theory and appliations*. New Jersey [u.a.] : World Scientific, 2006. – XIX, 641 S. : Ill., graph. Darst... – ISBN 9812566813 981–256–681–3 ; 9812566821

[Mel07] MELZER, Ingo: *Service-orientierte Architekturen mit Web Services : Konzepte - Standards - Praxis*. 2. Aufl. Heidelberg : Elsevier, Spektrum, Akad. Verl., 2007. – XXIX, 356 S. : graph. Darst., 25 cm. – ISBN 978–3–8274–1885–2

[ML01] MATTERN, Friedemann ; LANGHEINRICH, Marc: Allgegenwärtigkeit des Computers – Datenschutz in einer Welt intelligenter Alltagsdinge. In: MÜLLER, G. (Hrsg.) ; REICHENBACH, M. (Hrsg.): *Sicherheitskonzepte für das Internet*. Springer-Verlag, 2001, S. 7–26

[Moo65] MOORE, G. E.: Cramming More Components onto Integrated Circuits. In: *Electronics* 38 (1965), April, Nr. 8, S. 114–117

[MR00] MUEHLEN, Michael zur ; ROSEMANN, Michael: Workflow-Based Process Monitoring and Controlling $\frac{3}{4}$ Technical and Organizational Issues. In: *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6*. Washington, DC, USA : IEEE Computer Society, 2000. – ISBN 0–7695–0493–0, S. 6032

[MS95] MANSOURI-SAMANI, Massoud: *Monitoring of Distributed Systems*, Imperial College London, Diss., December 1995

[Neu94] In: NEUMAN, Clifford B.: *Scale in distributed systems*. Los Alamitos, CA : IEEE Computer Society, 1994, S. 463–489

[NFOP07] NEYEM, H. A. ; FRANCO, Rubén D. ; OCHOA, Sergio F. ; PINO, José A.: Supporting Mobile Workflow with Active Entities. In: SHEN, Weiming (Hrsg.) ; YANG, Yun (Hrsg.) ; YONG, Jianming (Hrsg.) ; HAWRYSZKIEWYCZ, Igor (Hrsg.) ; LIN, Zongkai (Hrsg.) ; BARTHES, Jean-Paul A. (Hrsg.) ; MAHER, Mary L. (Hrsg.) ; HAO, Qi (Hrsg.) ; TRAN, Minh H. (Hrsg.): *CSCWD*, IEEE, 2007, S. 795–800

[Nis08] NISSEN, Volker: *Service-orientierte Architekturen : Chancen und Herausforderungen bei der Flexibilisierung und Integration von Unternehmensprozessen*. Wiesbaden : Dt. Univ.-Verl., 2008. – VIII, 228 S. : Ill., graph. Darst.. – ISBN 978–3–8350–0815–1

[PA00] PAXSON, V. ; ALLMAN, M.: *Computing TCP's Retransmission Timer*. United States, 2000

[Pap03] PAPAZOGLOU, M.P.: Service-oriented computing: concepts, characteristics and directions. In: *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on* (10-12 Dec. 2003), S. 3–12

[PRD00] PERKINS, Charles E. ; ROYER, Elizabeth M. ; DAS, Samir R.: *IP Address Autoconfiguration for Ad Hoc Networks*. 2000

[RBB03] ROTHERMEL, Kurt ; BAUER, Martin ; BECKER, Christian: *Digitale Weltmodelle - Grundlage kontextbezogener Systeme*. Berlin [u.a.], 2003

[Ruf94] RUFFIN, Michel: A Survey of Logging Uses. In: *INRIA (France), Tech. Rep. BROADCAST-36. Also available as Univ. of Glasgow (Scotland), Tech. Rep* 2 (1994), S. 94–82

[Sat96] SATYANARAYANAN, M.: Fundamental challenges in mobile computing. In: *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA : ACM Press, 1996. – ISBN 0897918002, S. 1–7

[Sch00] SCHILLER, Jochen: *Mobilkommunikation : Techniken für das allgegenwärtige Internet ; [WAP, WWW, mobiles Internet, drahtloses ATM]*. München, Boston [u.a.] : Addison-Wesley, 2000 (Net.com). – 557 S. : Ill., graph. Darst., 25 cm. – ISBN 3–8273–1578–6

[Sch03] SCHMEES, Markus: Distributed digital commerce. In: *ICEC*, 2003, S. 131–137

[SGF02] SCHOLLMEIER, Rüdiger ; GRUBER, Ingo ; FINKENZELLER, Michael: Routing in mobile ad hoc and peer-to-peer networks: a comparison. In: *In Int.Workshop on Peer-to-Peer Computing. In Networking 2002*, 2002

[Sim08] SIMON, Walter: *GABALs großer Methodenkoffer - Managementtechniken*. 1. Aufl. Offenbach am Main : GABAL, 2008. – 336 S., 210 mm x 145 mm, 500 gr.. – ISBN 978–3–89749–901–0

[SS94] SINGHAL, Mukesh ; SHIVARATRI, Niranjan G.: *Advanced Concepts in Operating Systems*. New York, NY, USA : McGraw-Hill, Inc., 1994. – ISBN 007057572X

[SS07] SCHILL, Alexander ; SPRINGER, Thomas: *Verteilte Systeme : Grundlagen und Basistechnologien*. 1. Aufl. Berlin ; Heidelberg ; New York : Springer, 2007 (eXamen.press). – XIII, 368 S. : graph. Darst.. – ISBN 978–3–540–20568–5

[Str04] STRANG, Thomas: *Service-Interoperabilität in Ubiquitous-computing-Umgebungen*. Berlin : VDE-Verl., 2004 (Forschungs-Report). – 168 S. : graph. Darst.. – ISBN 3–8007–2823–0

[Tan03] TANENBAUM, Andrew S.: *Computer networks*. 4. ed. Upper Saddle River, NJ : Prentice Hall PTR, 2003. – XX, 891 S. : Ill., graph. Darst. ; 25cm. – ISBN 0–13–066102–3

[TP04] TUROWSKI, Klaus ; POUSTTCHI, Key: *Mobile Commerce : Grundlagen und Techniken ; mit 9 Tabellen*. Berlin [u.a.] : Springer, 2004. – IX, 224 S. : Ill., graph. Darst.. – ISBN 3–540–00535–8

[TS08] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme : Prinzipien und Paradigmen*. 2., aktualisierte Aufl.. München [u.a.] : Pearson Studium, 2008. – 761 S. : Ill., zahlr. graph. Darst.. – ISBN 978–3–8273–7293–2 ; 3–8273–7293–3

[Web98] WEBER, Michael: *Verteilte Systeme*. Heidelberg ; Berlin : Spektrum, Akad. Verl., 1998 (Spektrum-Hochschultaschenbuch). – XIV, 376 S. : graph. Darst.. – ISBN 3–8274–0221–2

[Wei95] WEISER, Mark: The computer for the 21st century. (1995), S. 933–940. ISBN 1558602461

[WK01] WANG, Helen J. ; KATZ, Randy H.: Mobility support in unified communication networks. In: *WOWMOM '01: Proceedings of the 4th ACM international workshop on Wireless mobile multimedia*. New York, NY, USA : ACM, 2001. – ISBN 1–58113–384–7, S. 95–102

[Wor98] WORKFLOW MANAGEMENT COALITION (Hrsg.): *Workflow Audit Trail (Interface 5) Application Programming Interface Specification Document Number: WfMC-TC-1022*. Workflow Management Coalition, 1998

[XT06] XYLOMENOS, George ; TSILOPOULOS, Christos: Adaptive Timeout Policies for Wireless Links. In: *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2466–4–01, S. 497–502

[ZK07] ZAPLATA, Sonja ; KUNZE, Christian P.: *Prozessmanagement im Mobile Computing - Kooperative Ausführung von Geschäftsprozessen im Umfeld serviceorientierter Architekturen*. VDM Verlag Dr. Müller, 2007. – ISBN: 978-3-8364-1010-6

[ZM04] ZUR MÜHLEN, Michael.: *Workflow-based process controlling : foundation, design, and application of workflow driven process information systems*. Berlin : Logos-Verl., 2004 (Advances in information systems and management science ; 6 ; Advances in information systems and management science ; 6). – XIV, 282 S. : graph. Darst. ; 24 cm.. – ISBN 3–8325–0388–9

[ZN06] ZHENG, Pei ; NI, Lionel: *Smart phone and next-generation mobile computing*. Amsterdam [u.a] : Elsevier/Morgan Kaufmann, 2006 (The Morgan Kaufmann series in networking). – XXVII, 551 S. : Ill., graph. Darst. ; 24cm.. – ISBN 0120885603 ; 9780120885602

[Zwa05] ZWAHR, Annette: *Der Grosse Brockhaus in einem Band : [mit CD-ROM ; mit beeindruckendem thematischem Karten- und Satellitenbildteil]*. 2. Aufl. Leipzig [u.a.] : Brockhaus, 2005. – 1175 S. : zahlr. Ill., graph. Darst., Kt. + 1 CD–ROM ; 12 cm. – ISBN 3–7653–3142–2 Gb

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, Juli 2009

Benjamin Böhling